

ADSP-214xx SHARC® Processor

Hardware Reference

***Includes ADSP-2146x, ADSP-2147x,
ADSP-2148x Product Families***

Revision 0.3, July 27, 2010

Part Number
82-000469-01

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



Copyright Information

© 2010 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, 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

CONTENTS

PREFACE

Purpose of This Manual	lxi
Intended Audience	lxi
Manual Contents	lxii
What's New in This Manual	lxv
Technical or Customer Support	lxvi
Registration for MyAnalog.com	lxvi
EngineerZone	lxvii
Social Networking Web Sites	lxvii
Supported Processors	lxvii
Product Information	lxviii
Analog Devices Web Site	lxviii
VisualDSP++ Online Documentation	lxviii
Technical Library CD	lxix
Notation Conventions	lxx

INTRODUCTION

Design Advantages	1-1
-------------------------	-----

Contents

SHARC Family Product Offerings	1-2
Processor Architectural Overview	1-2
Processor Core	1-2
I/O Peripherals	1-3
I/O Processor	1-3
Digital Audio Interface (DAI)	1-3
Interrupt Controller	1-4
Signal Routing Unit	1-4
Digital Peripheral Interface (DPI)	1-4
Interrupt Controller	1-4
Signal Routing Unit 2	1-5
Development Tools	1-5
Differences from Previous Processors	1-5
I/O Architecture Enhancements	1-6

I/O PROCESSOR

Features	2-2
Register Overview	2-3
DMA Channel Registers	2-3
DMA Channel Allocation	2-3
Standard DMA Parameter Registers	2-4
Extended DMA Parameter Registers	2-8
Data Buffers	2-10
Chain Pointer Registers	2-11
TCB Storage	2-13

Serial Port TCB	2-14
SPI TCB	2-14
UART TCB	2-15
Link Port TCB	2-15
FIR Accelerator TCB	2-16
IIR Accelerator TCB	2-17
FFT Accelerator TCB	2-18
External Port TCB	2-19
Clocking	2-22
Functional Description	2-22
Automated Data Transfer	2-22
DMA Transfer Types	2-23
DMA Direction	2-24
Internal to External Memory	2-24
Peripheral to Internal Memory	2-24
Peripheral to External Memory (SPORTs)	2-25
Internal Memory to Internal Memory	2-25
DMA Controller Addressing	2-25
Internal Index Register Addressing	2-27
External Index Register Addressing	2-28
DMA Channel Status	2-28
DMA Start and Stop Conditions	2-29
Operating Modes	2-30
DMA Chaining	2-32

Contents

TCB Memory Storage	2-32
Chain Assignment	2-33
Starting Chain Loading	2-34
TCB Chain Loading Priority	2-35
Chain Insert Mode (SPORTs Only)	2-36
Fixed DMA Channel Arbitration	2-36
Peripheral DMA Bus	2-42
External Port DMA Bus	2-43
SPORT/External Port DMA Bus	2-43
Rotating DMA Channel Arbitration	2-44
Rotating Priority by Group	2-44
Interrupts	2-45
Sources	2-45
Unchained DMA Interrupts	2-45
Chained DMA Interrupts	2-46
Transfer Completion Types	2-46
Internal Transfer Completion	2-47
Access Completion	2-47
Core Single Word Transfer Interrupts	2-47
Interrupt Versus Channel Priorities	2-48
Debug Features	2-49
Emulation Considerations	2-49
Effect Latency	2-49
Write Effect Latency	2-49

IOP Effect Latency	2-49
IOP Throughput	2-50
Programming Model	2-50
General Procedure for Configuring DMA	2-51

EXTERNAL PORT

Features	3-2
Pin Descriptions	3-3
Pin Multiplexing	3-3
Register Overview	3-3
Clocking AMI/SDRAM	3-5
Clocking AMI/DDR2	3-6
External Port Arbitration	3-7
Functional Description	3-7
Operating Mode	3-10
Arbitration Freezing	3-10
Asynchronous Memory Interface	3-10
Features	3-11
Functional Description	3-11
Asynchronous Reads	3-12
Asynchronous Writes	3-12
Parameter Timing	3-14
Idle Cycles	3-14
Address Mapping	3-14
Operating Modes	3-15

Contents

Data Packing	3-15
External Access Extension	3-15
Predictive Reads	3-16
SDRAM Controller	
(ADSP-2147x/ADSP-2148x)	3-17
Features	3-17
Functional Description	3-18
SDRAM Commands	3-20
Load Mode Register	3-20
Bank Activation	3-21
Single Precharge	3-21
Precharge All	3-22
Read/Write	3-22
Auto-Refresh	3-24
No Operation/Command Inhibit	3-24
Command Truth Table	3-24
Address Mapping	3-25
Address Translation Options	3-26
Address Width Settings	3-27
16-Bit Address Mapping	3-28
Refresh Rate Control	3-32
Internal SDRAM Bank Access	3-34
Single Bank Access	3-34
Multibank Access	3-35
Multi Bank Operation with Data Packing	3-36

Timing Parameters	3-37
Fixed Timing Parameters	3-37
Data Mask	3-38
Resetting the Controller	3-38
Operating Modes	3-38
Parallel Connection of SDRAMs	3-38
Buffering Controller for Multiple SDRAMs	3-39
SDRAM Read Optimization	3-40
Core Accesses	3-41
DMA Access	3-43
Notes on Read Optimization	3-43
Self-Refresh Mode	3-44
Forcing SDRAM Commands	3-45
Force Precharge All	3-45
Force Load Mode Register	3-46
Force Auto-Refresh	3-46
DDR2 DRAM Controller (ADSP-2146x)	3-46
Features	3-46
Pin Descriptions	3-48
Functional Description	3-48
DDR2 Commands	3-51
Load Mode Register	3-51
Load Extended Mode Register	3-52
Load Extended Mode Register 2	3-53

Contents

Load Extended Mode Register 3	3-53
Bank Activation	3-53
Precharge	3-54
Precharge All	3-54
Burst Read	3-54
Burst Write	3-55
Auto-Refresh	3-56
Self-Refresh Entry	3-57
Self-Refresh Exit	3-57
Precharge Power-Down Entry	3-58
Precharge Power-down Exit	3-59
No Operation/Command Inhibit	3-60
Address Mapping	3-60
Address Translation Options	3-60
Page Interleaving Map	3-60
Bank Interleaving Map	3-61
Address Width Settings	3-61
16-Bit Address Mapping	3-62
Address Map Tables	3-63
Refresh Rate	3-66
Data Mask	3-68
Resetting the Controller	3-68
Disabling the Controller	3-69
Initialization Sequence	3-69

Initialization Time	3-70
Internal DDR2 Bank Access	3-71
Single Bank Access	3-71
Multibank Access	3-71
Force Activation Window	3-72
Multi Bank Operation with Data Packing	3-73
Fixed Timing Parameters	3-74
Operating Modes	3-75
Parallel Connection of DDR2s	3-75
Buffering Controller for Multiple DDR2s	3-76
Read Optimization	3-76
DDR2 Read Optimization	3-77
Self-Refresh Mode	3-79
Single-Ended Data Strobe	3-81
On Die Termination (ODT)	3-81
Additive Latency	3-82
Forcing DDR2 Commands	3-82
Data Transfer	3-84
Data Buffers	3-84
AMI Receive Buffer	3-84
AMI Transmit Buffer	3-84
DMA Buffer	3-85
Core Access	3-85
External Port Dual Data Fetch	3-85

Contents

Conditional Instructions	3-86
SIMD Access	3-86
SDRAM	3-86
DDR2	3-87
External Instruction Fetch	3-88
Interrupt Vector Table (IVT)	3-88
Fetching ISA Instructions From External Memory	3-89
Instruction Packing	3-90
16-Bit Instruction Storage and Packing	3-90
8-Bit Instruction Storage and Packing	3-92
Mixing Instructions and Data in External Bank 0	3-93
Addressing for Various Memory Sizes	3-93
Writing Instructions to External Memory	3-94
Instruction Cache	3-95
Fetching VISA Instructions From External Memory	3-98
External Port DMA	3-100
External Port DMA Parameter Registers	3-100
Operating Modes	3-102
Internal DMA Addressing	3-102
Standard DMA	3-103
Circular Buffered DMA	3-103
Chained DMA Mode	3-104
Changing DMA Direction on the Fly	3-104
Scatter/Gather DMA	3-106

External Address Calculation	3-106
Delay Line DMA	3-111
External Address Calculation for Reads	3-112
Interrupts	3-114
Access Completion	3-115
Internal Transfer Completion	3-115
Interrupt Dependency on DMA Mode	3-115
External Port Throughput	3-116
AMI Data Throughput	3-117
SDRAM Throughput	3-117
Throughput Conditional Instructions	3-117
DDR2 Throughput	3-118
DMA Throughput	3-118
Core Throughput	3-118
External Instruction Fetch Throughput	3-119
Effect Latency	3-120
Write Effect Latency	3-121
Programming Models	3-121
External Port	3-121
DMA	3-121
Standard DMA	3-121
Chained DMA	3-122
Delay Line DMA	3-123
Disabling and Re-enabling DMA	3-124

Contents

Additional Information	3-124
AMI Initialization	3-125
SDRAM Controller	3-126
Power-Up Sequence	3-126
Output Clock Generator Programming Model	3-127
Self-Refresh Mode	3-127
Changing the VCO Clock During Runtime	3-128
DDR2 Controller	3-129
Power-Up Sequence	3-129
Frequency Change in Precharge Power-Down Mode	3-130
External Instruction Fetch	3-131
AMI Configuration	3-132
SDRAM Configuration	3-132
External Memory Access Restrictions	3-132

LINK PORTS—ADSP-2146X

Features	4-2
Pin Descriptions	4-3
Register Overview	4-3
Clocking	4-4
Functional Description	4-4
Architecture	4-5
Protocol	4-5
Intercommunication	4-7
Self-Synchronization	4-10

Multi-Master Conflicts	4-10
Example Token Passing	4-11
Data Transfer	4-13
Link Buffers	4-13
Transmit Buffer	4-14
Receive Buffer	4-14
Buffer Status	4-15
Core Transfers	4-15
DMA Transfers	4-16
Interrupts	4-16
Interrupt Sources	4-17
Interrupt Service	4-17
Access Completion	4-18
Internal Transfer Completion	4-18
DMA Access	4-19
Chained DMA	4-19
Core Access	4-19
Service Request Interrupts	4-20
Debug Features	4-21
Shadow Register	4-21
Buffer Hang Disable (BHD)	4-22
Effect Latency	4-22
Write Effect Latency	4-22
Link Port Effect Latency	4-22

Contents

Programming Model	4-22
Changing the Link Port Clock	4-23
Receive DMA	4-24
Transmit DMA	4-24

MEMORY-TO-MEMORY PORT DMA

Features	5-2
Register Overview	5-2
Clocking	5-2
Functional Description	5-3
Data Transfer Types	5-3
Data Buffer	5-3
DMA Transfer	5-4
Interrupts	5-4
MTM Throughput	5-5
Effect Latency	5-5
Write Effect Latency	5-5
MTM Effect Latency	5-5
Programming Model	5-5

FFT/FIR/IIR HARDWARE MODULES

FFT Accelerator	6-3
Features	6-4
Register Descriptions	6-4
Clocking	6-5

Functional Description	6-5
Compute Block	6-5
Data Memory	6-6
Coefficient Memory	6-6
Accelerator States	6-6
Reset State	6-6
Idle State	6-7
Read State	6-7
Processing State	6-7
Write State	6-7
Internal Memory Storage	6-8
Small FFT N<=256	6-8
Large FFT N>=256	6-9
Operating Modes	6-11
Small FFT Computation (<= 256 Points)	6-11
Large FFT Computation (>= 512 Points)	6-11
Example for FFT Size N=512	6-12
Vertical FFT	6-12
Special Product—Number of Iterations is N/128 = 4	6-12
Horizontal FFT	6-13
No Repeat Mode	6-14
Repeat Mode	6-14
Unpacked Data Mode	6-14
Inverse FFT	6-15

Contents

Data Transfer	6-15
FFT Buffers	6-15
DMA Transfers	6-15
DMA Channels and TCB Structure	6-16
Chained DMA	6-16
Interrupts	6-17
Interrupt Sources	6-18
Servicing DMA Interrupts	6-18
Servicing MAC Status Interrupts	6-18
FFT Performance	6-19
Small FFT (N is <= 256)	6-19
Large FFT (N >= 256)	6-19
Vertical FFT cycles	6-20
Special Prod cycles	6-20
Horizontal FFT cycles	6-20
Debug Features	6-20
Local Memory Access	6-20
Shadow Register	6-20
Effect Latency	6-21
Write Effect Latency	6-21
FFT Accelerator Effect Latency	6-21
Programming Model	6-21
N <= 256, No Repeat	6-21
N <= 256, Repeat	6-22

N >= 512, No Repeat	6-24
Configure the FFT Control Register	6-24
Vertical FFT Configuration	6-25
Special Buffer Configuration	6-25
Horizontal FFT Configuration	6-26
N >= 512, Repeat	6-26
Debug Mode	6-27
Write to Local Memory	6-27
Read from Local Memory	6-28
FIR Accelerator	6-28
Features	6-28
Register Overview	6-29
Clocking	6-29
Functional Description	6-30
Compute Block	6-31
Partial Sum Register	6-32
Delay Line Memory	6-33
Coefficient Memory	6-33
Prefetch Data Buffer	6-33
Processing Output	6-34
Internal Memory Storage	6-35
Coefficients and Input Buffer Storage	6-35
Operating Modes	6-37
Single Rate Processing	6-37

Contents

Single Iteration	6-37
Multi-Iteration	6-37
Window Processing	6-38
Multi Rate Processing	6-38
Decimation	6-38
Interpolation	6-39
Channel Processing	6-40
Floating-Point Data Format	6-42
Fixed-Point Data Format	6-42
Data Transfer	6-42
DMA Access	6-42
Chain Pointer DMA	6-43
Interrupts	6-44
Interrupt Sources	6-45
Debug Features	6-45
Local Memory Access	6-45
Single Step Mode	6-46
Emulation Considerations	6-46
Effect Latency	6-46
Write Effect Latency	6-46
FIR Accelerator Effect Latency	6-46
FIR Throughput	6-47
Additional Information	6-48
Programming Model	6-48

Single Channel Processing	6-48
Multichannel Processing	6-49
Debug Mode	6-52
Write to Local Memory	6-52
Read from Local Memory	6-52
Single Step Mode	6-53
FIR Programming Example	6-53
IIR Accelerator	6-55
Features	6-55
Register Overview	6-55
Clocking	6-56
Functional Description	6-56
Multiply and Accumulate (MAC) Unit	6-59
Data Memory	6-60
Coefficient Memory	6-60
Internal Memory Storage	6-60
Coefficient Memory Storage	6-60
Operating Modes	6-61
Window Processing	6-61
40-Bit Floating-Point Mode	6-61
Data Transfers	6-62
DMA Access	6-62
Chain Pointer DMA	6-62
Interrupts	6-64

Contents

Interrupt Sources	6-64
Debug Features	6-65
Local Memory Access	6-65
Single Step Mode	6-66
Emulation Considerations	6-66
Effect Latency	6-66
Write Effect Latency	6-67
IIR Accelerator Effect Latency	6-67
IIR Throughput	6-67
Programming Model	6-68
Writing to Local Memory	6-69
Reading from Local Memory	6-70
Single Step Mode	6-71
Programming Example	6-71

PULSE WIDTH MODULATION

Features	7-2
Pin Descriptions	7-4
Multiplexing Scheme	7-4
SRU Programming	7-5
Register Overview	7-5
Clocking	7-6
Functional Description	7-6
Two-Phase PWM Generator	7-6
Switching Frequencies	7-6

Duty Cycles	7-7
Dead Time	7-12
Output Control Unit	7-13
Output Enable	7-13
Output Polarity	7-13
Complementary Outputs	7-14
Crossover	7-14
Emergency Dead Time for Over Modulation	7-15
Output Control Feature Precedence	7-17
Operation Modes	7-18
Waveform Modes	7-18
Edge-Aligned Mode	7-18
Center-Aligned Mode	7-19
PWM Timer Edge Aligned Update	7-21
Single Update Mode	7-22
Double Update Mode	7-22
Effective Accuracy	7-23
Synchronization of PWM Groups	7-24
Interrupts	7-25
Debug Features	7-27
Status Debug Register	7-27
Emulation Considerations	7-27
Effect Latency	7-27
Write Effect Latency	7-27

Contents

PWM Effect Latency	7-27
--------------------------	------

MEDIA LOCAL BUS

Features	8-3
Pin Descriptions	8-3
Register Overview	8-4
Device Configuration and Status Registers	8-4
Channel Configuration Registers	8-4
Clocking	8-5
Functional Description	8-5
Operating Modes	8-7
Streaming Channel Frame Synchronization	8-7
Big-Endian and Little-Endian Mode	8-8
Data Transfer	8-8
Core Driven Data Transfer	8-8
I/O Local Channel Buffering	8-8
DMA	8-10
Ping-Pong DMA	8-11
Circular Buffer DMA	8-13
Interrupts	8-14
Interrupt Source	8-15
Servicing Interrupts	8-15
Debug Features	8-15
Loop-Back Test Mode	8-16
Programming Model	8-16

I/O Interrupt Mode	8-16
DMA Modes	8-17

DIGITAL APPLICATION/DIGITAL PERIPHERAL INTERFACES

Features	9-2
Register Overview	9-3
Clocking	9-4
Functional Description	9-4
DAI/DPI Signal Naming Conventions	9-7
I/O Pin Buffers	9-7
Pin Buffers as Signal Output	9-8
Pin Buffers as Signal Input	9-10
Pin Buffers as Open Drain	9-11
DAI/DPI Pin Buffer Status	9-11
Unused DAI/DPI Pins	9-12
Miscellaneous Buffers	9-12
DAI/DPI Peripherals	9-14
Output Signals With Pin Buffer Enable Control	9-14
Output Signals Without Pin Buffer Enable Control	9-16
Signal Routing Units (SRUs)	9-16
Signal Routing Matrix by Groups	9-16
DAI/DPI Group Routing	9-18
Rules for SRU Connections	9-20
Making SRU Connections	9-20

Contents

DAI Routing Capabilities	9-24
DPI Routing Capabilities	9-25
Pin Buffer Input	9-26
Pin Buffer Enable	9-26
Miscellaneous Signals	9-27
DAI Default Routing	9-28
DPI Default Routing	9-31
Interrupts	9-32
System Versus Exception Interrupts	9-32
Functional Description	9-33
DAI Interrupt Channels	9-33
DAI Interrupt Priorities	9-34
DPI Interrupt Channels	9-34
DPI Interrupt Priorities	9-34
DAI Miscellaneous Interrupts	9-35
DPI Miscellaneous Interrupts	9-35
DAI/DPI Interrupt Mask Events	9-36
DAI Interrupt Acknowledge	9-38
DPI Interrupt Acknowledge	9-39
Core versus DAI/DPI Interrupts	9-39
Debug Features	9-40
DAI Shadow Registers	9-40
DPI Shadow Registers	9-40
Loop Back Routing	9-40

Effect Latency	9-42
Write Effect Latency	9-42
Signal Routing Unit Effect Latency	9-42
Programming Model	9-42
DAI Example System	9-43

SERIAL PORTS

Features	10-2
Pin Descriptions	10-4
SRU Programming	10-5
SRU SPORT Receive Master	10-6
SRU SPORT Signal Integrity	10-6
Register Overview	10-7
Clocking	10-8
Master Clock	10-8
Master Frame Sync	10-9
Slave Mode	10-10
Functional Description	10-10
Architecture	10-11
Data Types and Companding	10-12
Companding the Data Stream	10-13
Transmit Path	10-14
Receive Path	10-15
Frame Sync	10-16
Sampling Edge	10-16

Contents

Frame Sync and Data Sampling	10-16
Serial Word Length	10-18
Internal Versus External Frame Syncs	10-18
External Frame Sync Sampling	10-19
Logic Level Frame Syncs	10-20
Data-Independent Frame Sync	10-20
Operation Modes	10-21
Mode Selection	10-23
Channel Order First	10-24
Standard Serial Mode	10-25
Timing Control Bits	10-25
Clocking Options	10-26
Frame Sync Options	10-26
Framed Versus Unframed Frame Syncs	10-26
Early Versus Late Frame Syncs	10-27
Left-Justified Mode	10-28
Master Serial Clock and Frame Sync Rates	10-29
Timing Control Bits	10-29
I ² S Mode	10-30
Master Serial Clock and Frame Sync Rates	10-30
Timing Control Bits	10-30
Multichannel Mode	10-31
Clocking Options	10-32
Frame Sync Options	10-32

Frame Sync Delay (MFD)	10-33
Transmit Data Valid Signal	10-33
Transmit Data Valid Output	10-34
Timing Control Bits	10-35
Number of Channels (NCH)	10-35
Active Channel Selection Registers	10-36
Companding Selection	10-36
Companding Limitations (ADSP-2146x)	10-37
Packed Mode	10-37
Clocking Options	10-38
Frame Sync Options	10-38
Timing Control Bits	10-39
Data Transfers	10-39
Data Buffers	10-40
Transmit Buffers (TXSPxA/B)	10-40
Receive Buffers (RXSPxA/B)	10-41
Buffer Status	10-41
Data Buffer Packing	10-42
Core Transfers	10-43
Single Word Transfers	10-43
Frame Sync Generation	10-44
Internal Memory DMA Transfers	10-44
External Memory DMA Transfers	10-45
Standard DMA	10-46

Contents

DMA Chaining	10-47
DMA Chain Insertion Mode	10-48
Frame Sync Generation	10-48
Interrupts	10-49
Internal Transfer Completion	10-50
Shared Channels	10-50
Error Detection	10-51
Error Status	10-53
Debug Features	10-53
SPORT Loopback	10-54
LoopBack Routing	10-54
Buffer Hang Disable (BHD)	10-54
Effect Latency	10-54
Write Effect Latency	10-55
SPORT Effect Latency	10-55
Programming Model	10-55
Setting Up and Starting DMA Master Mode	10-55
Setting Up and Starting Chained DMA	10-56
Enter DMA Chain Insertion Mode	10-56
Setting Up and Starting Multichannel Mode	10-57
Multichannel Mode Backward Compatibility	10-58
Programming Packed Mode	10-59
Additional Information for External Frame Sync Operation	10-59
Companding As a Function	10-60

INPUT DATA PORT

Features	11-2
Pin Descriptions	11-3
SRU Programming	11-5
Register Overview	11-5
Clocking	11-6
Functional Description	11-6
Operating Modes	11-8
PDAP Port Selection	11-9
Data Hold	11-9
PDAP Data Masking	11-10
PDAP Data Packing	11-10
No Packing	11-10
Packing by 2	11-11
Packing by 3	11-12
Packing by 4	11-13
Data Transfer	11-14
Data Buffer	11-14
Core Transfers	11-15
SIP Data Buffer Format	11-16
PDAP Data Buffer Format	11-18
DMA Transfers	11-19
Data Buffer Format for DMA	11-19
DMA Channel Priority	11-20

Contents

Standard DMA	11-20
Ping-Pong DMA	11-21
Multichannel DMA Operation	11-21
Multichannel FIFO Status	11-22
Interrupts	11-23
Interrupt Acknowledge	11-23
Threshold Interrupts	11-23
DMA Interrupts	11-24
FIFO Overflow Interrupts	11-24
Debug Features	11-25
Status register Debug	11-25
Buffer Hang Disable	11-25
Shadow Registers	11-25
Core FIFO Write	11-26
Effect Latency	11-26
Write Effect Latency	11-26
IDP Effect Latency	11-26
Programming Model	11-26
Setting Miscellaneous Bits	11-27
Starting Core Interrupt-Driven Transfer	11-27
Additional Notes	11-28
Starting A Standard DMA Transfer	11-29
Starting a Ping-Pong DMA Transfer	11-30
Serving Interrupts for DMA	11-31

ASYNCHRONOUS SAMPLE RATE CONVERTER

Features	12-2
Pin Descriptions	12-3
SRU Programming	12-3
Register Overview	12-4
Clocking	12-5
Functional Description	12-5
Serial Data Ports	12-9
Operating Modes	12-9
TDM Daisy Chain Mode	12-10
TDM Input Daisy Chain	12-11
TDM Output Daisy Chain	12-11
Bypass Mode	12-12
Matched-Phase Mode (ADSP-21488)	12-12
Data Format Matched-Phase Mode	12-14
Group Delay	12-14
Decimation Rate	12-15
Muting Modes	12-15
Soft Mute	12-15
Hard Mute	12-16
Auto Mute	12-16
Interrupts	12-16
Debug Features	12-17
Effect Latency	12-17

Contents

Write Effect Latency	12-17
SRC Effect Latency	12-18

SONY/PHILIPS DIGITAL INTERFACE

Features	13-2
Pin Descriptions	13-3
SRU Programming	13-4
Register Overview	13-6
Clocking	13-7
S/PDIF Transmitter	13-7
Functional Description	13-7
Input Data Format	13-9
Operating Modes	13-11
Full Serial Mode	13-11
Standalone Mode	13-11
Data Output Mode	13-12
S/PDIF Receiver	13-13
Functional Description	13-13
Clock Recovery	13-15
Output Data Format	13-15
Channel Status	13-16
Operating Modes	13-16
Compressed or Non-linear Audio Data	13-16
Emphasized Audio Data	13-17
Single-Channel Double-Frequency Mode	13-18

Clock Recovery Modes	13-18
Digital On-Chip PLL	13-18
External Analog PLL	13-19
Interrupts	13-19
Transmitter Interrupt	13-19
Receiver Interrupts	13-20
Receiver Error Interrupts	13-20
Debug Features	13-21
Loop Back Routing	13-21
Effect Latency	13-21
Write Effect Latency	13-21
Programming Model	13-21
Programming the Transmitter	13-21
Programming the Receiver	13-22
Interrupted Data Streams on the Receiver	13-23

PRECISION CLOCK GENERATOR

Features	14-2
Pin Descriptions	14-3
SRU Programming	14-4
Register Overview	14-5
Clocking	14-5
Functional Description	14-6
Serial Clock	14-6
Frame Sync	14-7

Contents

Frame Sync Output	14-7
Divider Mode Selection	14-8
Phase Shift	14-8
Pulse Width	14-9
Default Pulse Width	14-10
Timing Example for I2S Mode	14-11
Operating Modes	14-11
Normal Mode	14-12
Bypass Mode	14-13
One-Shot Mode	14-13
External Event Trigger	14-14
External Event Trigger Delay	14-15
Audio System Example	14-16
Clock Configuration Examples	14-18
Effect Latency	14-19
Write Effect Latency	14-19
PCG Effect Latency	14-19
Programming Model	14-20
Frame Sync Phase Setting	14-20
External Event Trigger	14-20
Debug Features	14-21

SERIAL PERIPHERAL INTERFACE PORTS

Features	15-2
Pin Descriptions	15-3

SRU Programming	15-4
Register Overview	15-5
Clocking	15-6
Choosing the Pin Enable for the SPI Clock	15-7
Functional Description	15-8
SPI Transaction	15-9
Single Master Systems	15-10
Multi Master Systems	15-11
Operating Modes	15-12
Transfer Initiate Mode	15-13
SPI Modes	15-14
Slave Select Outputs	15-15
Variable Frame Delay for Slave	15-17
Data Transfers	15-18
Buffers	15-18
Core Buffer Status	15-19
DMA Buffer Status	15-20
Core Transfers	15-20
Backward Compatibility	15-21
DMA Transfers	15-21
DMA Chaining	15-23
DMA Transfer Count	15-23
Full Duplex Operation	15-24
Interrupts	15-24

Contents

Interrupt Sources	15-24
Multi Master Error	15-26
Debug Features	15-27
Shadow Receive Buffers	15-27
Internal Loopback Mode	15-28
Loop Back Routing	15-28
Effect Latency	15-28
Write Effect Latency	15-29
SPI Effect Latency	15-29
Programming Model	15-29
Changing SPI Configuration	15-29
Master Mode Transfers	15-30
Core Master Transfers	15-31
DMA Master Transfers	15-31
Slave Mode Transfers	15-32
Core Slave Transfers	15-32
DMA Slave Transfers	15-33
Chained DMA Transfers	15-33
Stopping SPI Transfers	15-33
Switching From Transmit to a New DMA	15-34
Switching From Receive to a New DMA	15-35
DMA Error Interrupts	15-37
Multi-Master Transfers	15-38

PERIPHERAL TIMERS

Features	16-2
Pin Descriptions	16-3
SRU Programming	16-3
Register Overview	16-4
Read-Modify-Write	16-5
Clocking	16-5
Functional Description	16-5
Operating Modes	16-7
Pulse Width Modulation Mode (PWM_OUT)	16-8
PWM Waveform Generation	16-10
Single-Pulse Generation	16-11
Pulse Mode	16-12
Pulse Width Count and Capture Mode (WDTH_CAP)	16-12
External Event Watchdog Mode (EXT_CLK)	16-15
Interrupts	16-17
Sources	16-17
Watchdog Functionality	16-19
Debug Features	16-19
Loopback Routing	16-19
Loopback Routing	16-19
Effect Latency	16-19
Write Effect Latency	16-20
Peripheral Timers Effect Latency	16-20

Contents

Programming Model	16-21
PWM Out Mode	16-21
WDTH_CAP Mode	16-22
EXT_CLK Mode	16-23

SHIFT REGISTER – ADSP-2147X

Features	17-2
Pin Descriptions	17-3
SRU Programming	17-3
Register Overview	17-4
Clocking	17-4
Functional Description	17-5
Operating Modes	17-6
Serial Data Output	17-6
Parallel Data Output	17-7
Effect Latency	17-8
Write Effect Latency	17-8
Shift Register Effect Latency	17-8
Programming Model	17-8

REAL-TIME CLOCK—ADSP-2147X

Features	18-2
Pin Descriptions	18-3
Clocking	18-3
Register Overview	18-3

Functional Description	18-4
Interrupts	18-10

WATCHDOG TIMER – ADSP-2147X

Features	19-2
Pin Descriptions	19-3
Register Overview	19-3
Clocking	19-4
Functional Description	19-4
Operating Mode	19-6
Trip Count	19-6
Debug Features	19-7
Emulation Considerations	19-7
Effect Latency	19-7
Write Effect Latency	19-7
Watchdog Timers Effect Latency	19-7
Programming Model	19-7

UART PORT CONTROLLER

Features	20-2
SRU Programming	20-3
Register Overview	20-3
Clocking	20-4
Functional Description	20-5
Serial Communication	20-7

Contents

Operating Modes	20-8
Data Packing	20-8
9-Bit Transmission Mode	20-8
Packed Mode	20-9
Data Transfer Types	20-10
Data Buffers	20-10
Transmit Holding Registers (UARTTHR)	20-10
Receive Buffer Registers (UARTRBR)	20-11
Core Transfers	20-12
DMA Transfers	20-13
DMA Chaining	20-14
Interrupts	20-14
Interrupt Routing	20-15
DPI	20-15
UART	20-16
DMA Interrupts	20-16
Core Interrupts	20-17
Error Interrupts	20-19
Debug Features	20-20
Shadow Registers	20-20
Shadow Buffer	20-20
Loop Back Routing	20-20
Effect Latency	20-20
Write Effect Latency	20-21

UART Effect Latency	20-21
Programming Model	20-21
Autobaud Detection	20-21
Programming Model for DMA Transfers	20-22
Setting Up and Starting Chained DMA	20-22
Notes on Using UART DMA	20-23
Programming Model for Core Transfers	20-24

TWO WIRE INTERFACE CONTROLLER

Features	21-2
Pin Descriptions	21-3
SRU Programming	21-4
Clocking	21-4
Register Overview	21-5
Functional Description	21-6
Bus Arbitration	21-9
Start and Stop Conditions	21-10
Slave Mode Addressing	21-11
Master Mode Addressing	21-11
Data Transfer	21-12
Data Buffers	21-12
8-Bit Transmit FIFO Register	21-12
16-Bit Transmit FIFO Register	21-12
8-Bit Receive FIFO Register	21-13
16-Bit Receive FIFO Register	21-14

Contents

Operating Modes	21-14
General Call Addressing	21-14
Fast Mode	21-15
Interrupts	21-15
Interrupt Routing	21-16
DPI	21-16
TWI	21-16
Interrupt Sources	21-17
Debug Features	21-18
Buffer Hang Disable	21-18
Loop Back Routing	21-18
Effect Latency	21-19
Write Effect Latency	21-19
TWI Effect Latency	21-19
Programming Model	21-19
General Setup	21-19
Slave Mode	21-20
Master Mode Clock Setup	21-21
Master Mode Transmit	21-21
Master Mode Receive	21-22
Repeated Start Condition	21-24
Transmit/Receive Repeated Start Sequence	21-24
Receive/Transmit Repeated Start Sequence	21-25
Electrical Specifications	21-26

POWER MANAGEMENT

Features	22-1
Register Overview	22-1
Phase-Locked Loop (PLL)	22-2
Functional Description	22-2
PLL Input Clock	22-3
Pre-Divider Input	22-3
PLL Multiplier	22-4
PLLM Hardware Control	22-4
PLLM Software Control	22-4
PLL VCO	22-5
Output Clock Generator	22-5
Core Clock (CCLK)	22-6
IOP Clock (PCLK)	22-6
SDRAM/DDR2 Clock (SDCLK _x /DDR2_CLK)	22-6
Default PLL Hardware Settings	22-6
Operating Modes	22-7
Bypass Mode	22-7
Normal Mode	22-8
Clocking Golden Rules	22-8
Power-Up Sequence	22-8
PLL Start-Up	22-9
Power Management	22-10
Peripherals	22-10

Contents

DAI Routing Unit	22-10
External Port Control	22-11
Disabling the SDRAM Controller	22-11
Disabling the DDR2 Controller	22-11
Disconnect DAI/DPI Pin Buffers	22-12
Disable the S/PDIF Receiver	22-12
Example for Clock Management	22-12
General Notes on Power Savings	22-13
Programming Models	22-13
Post Divider	22-14
Multiplier and Post Divider Programming Model	22-15
Back to Back Bypass	22-17

SYSTEM DESIGN

Features	23-1
Pin Descriptions	23-2
Register Overview	23-2
Processor Reset	23-3
Hardware Reset	23-3
Software Reset	23-4
Running Reset	23-4
System Considerations	23-6
External Host	23-7
Processor Booting	23-7
Boot Mechanisms	23-8

External Port Booting	23-8
SPI Port Booting	23-12
Master Boot Mode	23-12
Master Header Information	23-14
Slave Boot Mode	23-15
SPI Boot Packing	23-17
32-Bit SPI Packing	23-18
16-Bit SPI Packing	23-19
8-Bit SPI Packing	23-20
Link Port Booting	23-21
Kernel Boot Time	23-23
ROM Booting	23-24
Programming Model	23-24
Running Reset	23-25
Running The Boot Kernel	23-25
Loading the Boot Kernel Using DMA	23-25
Executing the Boot Kernel	23-26
Loading the Application	23-26
Loading the Application's Interrupt Vector Table	23-26
Starting Program Execution	23-27
Memory Aliasing in Internal Memory	23-27
Pin Multiplexing	23-28
Core FLAG Pins Multiplexing	23-28
Backward Compatibility	23-29

Contents

External Port Pin Multiplexing	23-29
Multiplexed External Port Pins	23-30
Backward Compatibility	23-31
Parallel Connection of Flag Pins via External Port and DPI Pins	23-31
High Frequency Design	23-33
Circuit Board Design	23-33
Clock Input Specifications and Jitter	23-33
RESETOUT	23-34
Input Pin Hysteresis	23-34
Pull-Up/Pull-Down Resistors	23-35
Memory Select Pins	23-35
Edge-Triggered I/O	23-35
Asynchronous Inputs	23-36
Decoupling and Grounding	23-36
Circuit Board Layout	23-37
Other Recommendations and Suggestions	23-37
EZ-KIT Lite Schematics	23-39
Oscilloscope Probes	23-39
Recommended Reading	23-39
System Components	23-40
Power Management Circuits	23-40
Supervisory Circuits	23-41
Definition of Terms	23-43

REGISTERS REFERENCE

Overview	A-2
Register Diagram Conventions	A-2
Bit Types and Settings	A-3
System and Power Management Registers	A-4
System Control Register (SYSCTL)	A-4
ADSP-2146x Power Management Registers	A-6
Power Management Control Registers (PMCTL)	A-7
Power Management Control Register 1 (PMCTL1)	A-9
ADSP-2147x/ADSP-2148x Power Management Registers	A-12
Power Management Control Registers (PMCTL)	A-12
Power Management Control Register 1 (PMCTL1)	A-15
Running Reset Control Register (RUNRSTCTL)	A-18
ADSP-2146x External Port Registers	A-18
External Port Control Register (EPCTL)	A-18
AMI Control Registers (AMICTLx)	A-21
AMI Status Register (AMISTAT)	A-23
DDR2 Registers	A-24
DDR2 Control Register 0 (DDR2CTL0)	A-25
DDR2 Timing Control Register 1 (DDR2CTL1)	A-29
DDR2 Control Register 2 (DDR2CTL2)	A-31
DDR2 Control Register 3 (DDR2CTL3)	A-33
DDR2 Control Register 4 (DDR2CTL4)	A-35

Contents

DDR2 Control Register 5 (DDR2CTL5)	A-35
Refresh Rate Control Register (DDR2RRC)	A-36
Controller Status Register 0 (DDR2STAT0)	A-37
Controller Status Register 1 (DDR2STAT1)	A-39
DLL0 Control Register 1 (DLL0CTL1)	A-40
DLL1 Control Register 1 (DLL1CTL1)	A-41
DLL Status Registers (DLL0STAT0, DLL1STAT0)	A-42
DDR2 Pad Control Register 0 (DDR2PADCTL0)	A-43
DDR2 Pad Control Register 1 (DDR2PADCTL1)	A-44
ADSP-2147x, ADSP-2148x External Port Registers	A-45
External Port Control Register (EPCTL)	A-45
AMI Control Registers (AMICTL x)	A-47
AMI Status Register (AMISTAT)	A-51
SDRAM Registers	A-51
Control Register (SDCTL)	A-51
Control Status Register 0 (SDSTAT0)	A-55
Controller Status Register 1 (SDSTAT1)	A-57
Refresh Rate Control Register (SDRRC)	A-58
External Port DMA Control Registers (DMAC x)	A-60
Peripheral Registers	A-63
Link Port Registers	A-63
Control Register (LCTL x)	A-63
Status Registers (LSTAT x)	A-65
Memory-to-Memory Registers	A-66

DMA Control (MTMCTL Register)	A-66
Pulse Width Modulation Registers	A-67
Global Control Register (PWMGCTL)	A-67
Global Status Register (PWMGSTAT)	A-69
Control Register (PWMCTL _x)	A-69
Status Registers (PWMSTAT _x)	A-71
Output Disable Registers (PWMSEG _x)	A-71
Polarity Select Registers (PWMPOL _x)	A-72
Period Registers (PWMPERIOD _x)	A-73
Duty Cycle High Side Registers (PWMA _x , PWMB _x)	A-73
Duty Cycle Low Side Registers (PWMA _{Lx} , PWMB _{Lx})	A-74
Dead Time Registers (PWMDT _x)	A-74
Debug Status Registers (PWMDBG _x)	A-74
FFT Accelerator Registers	A-74
General Control Register (FFTCTL1)	A-75
Control Register (FFTCTL2)	A-76
Multiplier Status Register (FTTMACSTAT)	A-78
DMA Status Register	A-78
Debug Registers (FTTDADDR, FFTDDATA)	A-79
FIR Accelerator Registers	A-79
Global Control Register (FIRCTL1)	A-79
Channel Control Register (FIRCTL2)	A-81
FIR MAC Status Register (FIRMACSTAT)	A-83
FIR DMA Status Register (FIRDMASTAT)	A-85

Contents

FIR Debug Registers (FIRDEBUGCTL, FIRDBGADDR)	A-86
IIR Accelerator Registers	A-87
IIR Global Control Register (IIRCTL1)	A-87
IIR Channel Control Register (IIRCTL2)	A-90
IIR MAC Status Register (IIRMACSTAT)	A-91
IIR DMA Status Register (IIRDMASTAT)	A-91
IIR Debug Registers (IIRDEBUGCTL, IIRDEBUGADDR)	A-93
Media Local Bus Registers	A-94
MLB Global Registers	A-94
Device Control Configuration Register (MLB_DCCR) .	A-94
System Status Register (MLB_SSCR)	A-96
System Data Configuration Register (MLB_SDCR)	A-97
System Mask Configuration Register (MLB_SMCR)	A-98
Channel Interrupt Status Register (MLB_CICR)	A-99
MLB Base Registers	A-99
Logical Channel Registers	A-101
Channel Control Registers (MLB_CECRx)	A-102
Channel Status Configuration Registers (MLB_CSCRx)	A-108
Channel x Current Buffer Configuration Registers (MLB_CCBCRx)	A-111
Channel x Next Buffer Configuration Registers (MLB_CNBCRx)	A-112

Local Buffer Configuration Registers (MLB_LCBCRx)	A-112
Watchdog Timer Registers	A-114
Control (WDTCTL)	A-114
Status (WDTSTATUS)	A-114
Current Count (WDTCURCNT)	A-115
Trip Counter (WDTTRIP)	A-115
Clock Select (WDTCLKSEL)	A-116
Period (WDTCNT)	A-117
Unlock (WDTUNLOCK)	A-117
DAI Signal Routing Unit Registers	A-118
Clock Routing Control Registers (SRU_CLKx, Group A)	A-118
Serial Data Routing Registers (SRU_DATx, Group B)	A-123
Frame Sync Routing Control Registers (SRU_FSx, Group C)	A-128
Pin Signal Assignment Registers (SRU_PINx, Group D)	A-132
Miscellaneous Signal Routing Registers (SRU_MISCx, Group E)	A-138
DAI Pin Buffer Enable Registers (SRU_PBENx, Group F)	A-141
DAI Shift Register Routing Registers (Group G, ADSP-2147x)	A-145
Clock Routing Register (SRU_CLK_SHREG)	A-145
Data Routing Register (SRU_DAT_SHREG)	A-147

Contents

DAI Pin Buffer Registers	A-148
Pin Buffer Registers (DAI_PIN_STAT)	A-148
Interrupt Controller Registers	A-149
Peripherals Routed Through the DAI	A-150
Serial Port Registers	A-150
SPORT Divisor Registers (DIVx)	A-151
Serial Control Registers (SPCTLx)	A-151
SPORT Control 2 Registers (SPCTLNx)	A-167
SPORT Multichannel Control Registers (SPMCTLx)	A-169
SPORT Active Channel Select Registers (MTxCSSy or MRxCSSy)	A-171
SPORT Compand Registers (MTxCCSSy or MRxCCSSy) ..	A-172
Error Control Register (SPERRCTLx)	A-172
SPORT Error Status Register (SPERRSTAT)	A-174
Input Data Port Registers	A-174
Input Data Port DMA Control Registers	A-174
Input Data Port Control Register 0 (IDP_CTL0)	A-175
Input Data Port Control Register 1 (IDP_CTL1)	A-177
Input Data Port Control Register 2 (IDP_CTL2)	A-178
Parallel Data Acquisition Port Control Register (IDP_PP_CTL)	A-179
IDP Status Register (DAI_STAT0)	A-182
IDP Status Register 1 (DAI_STAT1)	A-183
Sample Rate Converter Registers	A-184
Control Registers (SRCCTLx)	A-184

Mute Register (SRCMUTE)	A-189
Ratio Registers (SRCRATx)	A-189
Precision Clock Generator Registers	A-191
Control Registers (PCG_CTLxy)	A-191
Clock Inputs	A-193
Pulse Width Registers (PCG_PWx)	A-194
PCG Frame Synchronization Registers (PCG_SYNCx) ...	A-196
Sony/Philips Digital Interface Registers	A-199
Transmitter Registers	A-199
Transmit Control Register (DITCTL)	A-199
Transmit Status Bit Registers for Subframe A/B (DITCHANAx/Bx)	A-202
Transmit User Bits Buffer Registers for Subframe A/B Registers (DITUSRBITAx/Bx)	A-203
User Bit Update Register (DITUSRUPD)	A-204
Receiver Registers	A-204
Receive Control Register (DIRCTL)	A-204
Receive Status Register (DIRSTAT)	A-206
Receive Status Registers for Subframe A (DIRCHAN)	A-209
Receive Status Registers for Subframe B (DIRCHANB)	A-209
Real-Time Clock Registers	A-209
Control Register (RTC_CTL)	A-210
Status Register (RTC_STAT)	A-211

Contents

Stopwatch Count Register (RTC_SWTCH)	A-213
Clock Register (RTC_CLOCK)	A-214
Alarm Register (RTC_ALARM)	A-214
Initialization Register (RTC_INIT)	A-215
Initialization Status Register (RTC_INITSTAT)	A-216
Shift Register Register	A-217
Control Register (SR_CTL)	A-217
DPI Signal Routing Unit Registers	A-218
Miscellaneous Signal Routing Registers (SRU2_INPUTx, Group A)	A-218
Pin Assignment Signal Routing (SRU2_PINx, Group B)	A-223
Pin Enable Signal Routing (SRU2_PBENx, Group C)	A-226
DPI Pin Buffer Registers	A-230
Pin Buffer Status Register (DPI_PIN_STAT)	A-230
DPI Interrupt Controller Registers	A-230
Peripherals Routed Through the DPI	A-231
Serial Peripheral Interface Registers	A-232
Control Registers (SPICTL, SPICTLB)	A-232
DMA Configuration Registers (SPIDMAC, SPIDMACB)	A-237
Baud Rate Registers (SPIBAUD, SPIBAUDB)	A-239
Status (SPISTAT, SPISTATB) Registers	A-240
SPI Port Flags Registers (SPIFLG, SPIFLGB)	A-242

UART Control and Status Registers	A-243
Line Control Register (UART0LCR)	A-243
Line Status Register (UART0LSR)	A-245
Interrupt Enable Register (UART0IER)	A-246
Interrupt Identification Registers (UART0IIR, UART0IIRSH)	A-247
Divisor Latch Registers (UART0DLL, UART0DLH)	A-249
Scratch Register (UART0SCR)	A-249
Mode Register (UART0MODE)	A-249
Buffer Control Registers (UART0TXCTL, UART0RXCTL)	A-251
DMA Status Registers (UART0TXSTAT, UART0RXSTAT)	A-252
Two Wire Interface Registers	A-253
Master Internal Time Register (TWIMITR)	A-253
Clock Divider Register (TWIDIV)	A-254
Slave Mode Control Register (TWISCTL)	A-255
Slave Address Register (TWISADDR)	A-256
Slave Status Register (TWISSTAT)	A-256
Master Control Register (TWIMCTL)	A-257
Master Address Register (TWIMADDR)	A-260
Master Status Register (TWIMSTAT)	A-260
FIFO Control Register (TWIFIFOCTL)	A-263
FIFO Status Register (TWIFIFOSTAT)	A-264
Interrupt Latch Register (TWIIRPTL)	A-265

Contents

Interrupt Enable Register (TWIIMASK)	A-267
Peripheral Timer Registers	A-269
Read-Modify-Write Timer Control Register	A-269
Timer Configuration Registers (TMxCTL)	A-270
Timer Status Registers (TMxSTAT)	A-271
Register Listing	A-273

PERIPHERAL INTERRUPT CONTROL

Interrupt Latency	B-1
Interrupt Acknowledge	B-2
Interrupt Completion	B-3
Interrupt Priority	B-4
Peripherals with Multiple Interrupt Vector Addresses	B-7
Priority Interrupt Control Registers (PICRx)	B-8

AUDIO FRAME FORMATS

Overview	C-2
Standard Serial Mode	C-2
I ² S Mode	C-3
Left-Justified Mode	C-5
Right-Justified Mode	C-5
TDM Mode	C-6
Packed I ² S Mode	C-7
MOST Mode	C-8
AES/EBU/SPDIF Formats	C-9

Subframe Format	C-12
Channel Coding	C-14
Preambles	C-15

INDEX

Contents

PREFACE

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

Purpose of This Manual

The *ADSP-214xx SHARC Processor Hardware Reference* contains information about the DSP architecture and DSP assembly language for these processors. These are 32-bit, fixed- and floating-point digital signal processors from Analog Devices for use in computing, communications, and consumer applications.

Intended Audience

The primary audience for this manual is a programmer who is familiar with Analog Devices processors. 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

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

- Chapter 1, “[Introduction](#)”
Provides an architectural overview of the SHARC processors.
- Chapter 2, “[I/O Processor](#)”
Describes input/output processor architecture, and provides direct memory access (DMA) procedures for the processor peripherals.
- Chapter 3, “[External Port](#)”
Describes how the processor’s connect to external memories. These include DDR2 (ADSP-2146x) and SDRAM (ADSP-2147x, ADSP-2148x).
- Chapter 4, “[Link Ports—ADSP-2146x](#)”
Describes the two bidirectional 8-bit wide link ports, which can connect to other processor or peripheral link ports.
- Chapter 5, “[Memory-to-Memory Port DMA](#)”
Describes memory-to-memory DMA.
- Chapter 6 “[FFT/FIR/IIR Hardware Modules](#)”
Describes the dedicated hardware accelerators used to reduce the instruction load on the core, freeing it up for other tasks, effectively adding more bandwidth.
- Chapter 7, “[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 8, “[Media Local Bus](#)”
Details the Media Local Bus port (MLB), an on-PCB or inter-chip communication bus, which allows an application to access MOST network data.
- Chapter 9, “[Digital Application/Digital Peripheral Interfaces](#)”
Provides information about the digital audio/digital peripheral interface (DAI/DPI) which allows you to attach an arbitrary number and variety of peripherals to the SHARC processor while retaining high levels of compatibility.
- Chapter 10, “[Serial Ports](#)”
Describes the 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 11, “[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 12, “[Asynchronous Sample Rate Converter](#)”
Provides information on the sample rate converter (SRC) module. This module performs synchronous or asynchronous sample rate conversion across independent stereo channels, without using any internal processor resources.
- Chapter 13, “[Sony/Philips Digital Interface](#)”
Provides information on the use of the Sony/Philips Digital Interface which is a standard audio file transfer format that allows the transfer of digital audio signals from one device to another without having to be converted to an analog signal.
- Chapter 14, “[Precision Clock Generator](#)”
Details the precision clock generators (PCG), each of which generates a pair of signals derived from a clock input signal.

Manual Contents

- Chapter 15, “Serial Peripheral Interface Ports”
Describes the operation of the serial peripheral interface (SPI) port. SPI devices communicate using a master-slave relationship and can achieve high data transfer rate because they can operate in full-duplex mode.
- Chapter 16, “Peripheral Timers”
Describes three identical 32-bit timers that can be used to interface with external devices.
- Chapter 17, “Shift Register – ADSP-2147x”
Describes the 18 stage serial in, serial/parallel out shift register.
- Chapter 18, “Real-Time Clock—ADSP-2147x”
Describes the digital watch features.
- Chapter 19, “WatchDog Timer – ADSP-2147x”
Describes software watchdog function which can improve system reliability by forcing the processor to a known state.
- Chapter 20, “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 21, “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 22, “System Design”
Describes system design features of the ADSP-214xx processors. These include power, reset, clock, JTAG, and booting, as well as pin multiplexing schemes and other system-level information.

- Chapter 23, “[Power Management](#)”
Describes system design features as they relate to power management.
- Appendix A, “[Registers Reference](#)”
Provides a graphical presentation of all registers and describes the bit usage in each register.
- Appendix B “[Peripheral Interrupt Control](#)”
Provides a complete listing of the registers that are used to configure and control interrupts.
- Appendix C “[Audio Frame Formats](#)”
Provides descriptions on the standard audio formats used by many of the peripherals.



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

What's New in This Manual

This is the third general release of the preliminary edition (Revision 0.3) of the this manual. This manual has been retitled from *ADSP-2146x SHARC Processor Hardware Reference* to *ADSP-214xx SHARC Processor Hardware Reference*. This change is due to the addition of two new product families—the ADSP-2147x and ADSP-2148x SHARC processors.

The ADSP-2147x and ADSP-2148x SHARC processors contain additional features. These features are listed in each product's respective data sheet.

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/technical_support
- 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

Registration for 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 about products you are interested in. Click **Register** to use this site. 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.

EngineerZone

EngineerZone is a technical support forum from Analog Devices. It allows you direct access to ADI technical support engineers. You can search FAQs and technical information to get quick answers to your embedded processing and DSP design questions.

Use EngineerZone to connect with other DSP developers who face similar design challenges. You can also use this open forum to share knowledge and collaborate with the ADI support team and your peers. Visit <http://ez.analog.com> to sign up.

Social Networking Web Sites

You can now follow Analog Devices SHARC development on Twitter and LinkedIn. To access:

- Twitter: <http://twitter.com/ADISHARC>
- LinkedIn: Network with the LinkedIn group, Analog Devices SHARC: <http://www.linkedin.com>

Supported 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, ADSP-2137x, ADSP-2146x, ADSP-2147x and ADSP-2148x.

Product Information

Product information can be obtained from the Analog Devices Web site, VisualDSP++ online Help system, and a technical library CD.

Analog Devices Web Site

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

To access a complete technical library for each processor family, go to http://www.analog.com/processors/technical_library. The manuals selection opens a list of current manuals related to the product as well as a link to the previous revisions of the manuals. When locating your manual title, note a possible errata check mark next to the title that leads to the current correction report against the manual.

Also note, [MyAnalog.com](http://www.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, including documentation errata against all manuals.

[MyAnalog.com](http://www.myanalog.com) provides access to books, application notes, data sheets, code examples, and more.

Visit www.myanalog.com to sign up. If you are already a registered user, just log on. Your user name is your e-mail address.

VisualDSP++ Online Documentation

Online documentation comprises the VisualDSP++ Help system, software tools manuals, hardware tools manuals, processor manuals, Dinkum Abridged C++ library, and FLEXnet License Tools software

documentation. You can search easily across the entire VisualDSP++ documentation set for any topic of interest.

For easy printing, supplementary Portable Documentation Format (.pdf) files for all manuals are provided on the VisualDSP++ installation CD.

Each documentation file type is described as follows.

File	Description
.chm	Help system files and manuals in Microsoft help format
.htm or .html	Dinkum Abridged C++ library and FLEXnet License Tools software documentation. Viewing and printing the .html files requires a browser, such as Internet Explorer 6.0 (or higher).
.pdf	VisualDSP++ and processor manuals in PDF format. Viewing and printing the .pdf files requires a PDF reader, such as Adobe Acrobat Reader (4.0 or higher).

Technical Library CD

The technical library CD contains seminar materials, product highlights, a selection guide, and documentation files of processor manuals, VisualDSP++ software manuals, and hardware tools manuals for the following processor families: Blackfin®, SHARC, TigerSHARC®, ADSP-218x, and ADSP-219x.

To order the technical library CD, go to <http://www.analog.com-processors/manuals>, navigate to the manuals page for your processor, click the request CD check mark, and fill out the order form.

Data sheets, which can be downloaded from the Analog Devices Web site, change rapidly, and therefore are not included on the technical library CD. Technical manuals change periodically. Check the Web site for the latest manual revisions and associated documentation errata.

Notation Conventions

Text conventions used in this manual are identified and described as follows. Note that additional conventions, which apply only to specific chapters, may appear throughout this document.

Example	Description
Close command (File menu)	Titles in reference sections indicate the location of an item within the VisualDSP++ environment's menu system (for example, the Close command appears on the File menu).
{this that}	Alternative items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as this or that . One or the other is required.
[this that]	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional this or that .
[this,...]	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of this .
.SECTION	Commands, directives, keywords, and feature names are in text with letter gothic font.
<i>filename</i>	Non-keyword placeholders appear in text with italic style format.
SWRST Software Reset register	Register names appear in UPPERCASE and a special typeface. The descriptive names of registers are in mixed case and regular typeface.
TMROE, RESET	Pin names appear in UPPERCASE and a special typeface. Active low signals appear with an OVERBAR .
DRx, I[3:0] SMS[3:0]	Register, bit, and pin names in the text may refer to groups of registers or pins: A lowercase x in a register name (DRx) indicates a set of registers (for example, DR2, DR1, and DRO). A colon between numbers within brackets indicates a range of registers or pins (for example, I[3:0] indicates I3, I2, I1, and I0; SMS[3:0] indicates SMS3, SMS2, SMS1, and SMS0).
0abcd, b#1111	A 0x prefix indicates hexadecimal; a b# prefix indicates binary.

Example	Description
	<p>Note: For correct operation, ...</p> <p>A Note: provides supplementary information on a related topic. In the online version of this book, the word Note appears instead of this symbol.</p>
	<p>Caution: Incorrect device operation may result if ...</p> <p>Caution: Device damage may result if ...</p> <p>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.</p>
	<p>Warning: Injury to device users may result if ...</p> <p>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.</p>

Notation Conventions

1 INTRODUCTION

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

Design Advantages

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

SHARC Family Product Offerings

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

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

Processor Architectural Overview

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

Processor Core

The processor core consists of two processing elements (each with three computation units and data register file), a program sequencer, two data address generators, a timer, and an instruction cache. Digital signal processing occurs primarily in the processor core. The FFT, FIR, and IIR accelerators each contain dedicated signal processing units to off load core processing for these units.

I/O Peripherals

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

- Asynchronous Memory Interface (AMI)
- SDRAM controller (ADSP-2147x, ADSP-2148x)
- DDR2 controller (ADSP-2146x)
- 4 PWM modules

I/O Processor

The input/output processor (IOP) manages the off-chip data I/O to free the core from this burden. Up to 67 channels of DMA are available on the processors. For model specific information, see the product specific data sheet. 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.

Digital Audio Interface (DAI)

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

- 8 serial ports (SPORT)
- Input Data Port (IDP)
- 4 precision clock generators (PCG)
- Some family members have an S/PDIF receiver/transmitter

Processor Architectural Overview

- 4 asynchronous sample rate converters (ASRC)
- DTCP encryption

Interrupt Controller

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

Signal Routing Unit

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

Digital Peripheral Interface (DPI)

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

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

Interrupt Controller

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

Signal Routing Unit 2

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

Development Tools

The 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-214xx processors and previous SHARC processors: ADSP-21161, ADSP-21160, ADSP-21060, ADSP-21061, ADSP-21062, and ADSP-21065L. Like the ADSP-2116x family, the ADSP-214xx SHARC processor family is based on the original ADSP-2106x SHARC family. The ADSP-214xx processors preserve much of the ADSP-2106x architecture and is code compatible to the ADSP-21160, while extending performance and functionality. For background information on 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 the ADSP-2116x processors. This architecture incorporates two independent DMA buses versus the previous SHARC DMA controllers:

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

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

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-214xx processors contain an I/O processor (IOP) that supports a variety of DMA (direct memory access) operations. Each DMA operation transfers an entire block of data. These operations include the transfer types shown in [Table 2-1](#) and the list that follows.

Table 2-1. I/O Processor Specifications

Feature	Availability
Total DMA channels	See product specific data sheet
Rotating DMA channel priority	Yes
Media Local Bus (MLB)	31
SPORT DMA channels	16
IDP DMA channels	8
UART DMA channels	4
FIR/FFT/IIR DMA channels	2
SPI DMA channels	2
MTM/DTCP DMA channels	2
External Port DMA channels	2
PDAP DMA channel	1
DMA channel interrupts	16
Clock Operation	f_{PCLK}

Features

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

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

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

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

Register Overview

Two global IOP registers control the DMA arbitration over the I/O buses—the first for the peripheral bus and the second for the external port bus. This section provides brief descriptions of the major IOP registers. For complete information, see “[Register Listing](#)” on page [A-273](#).

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

External Port Control Register (EPCTL). Controls the external port DMA operation for fixed or rotating DMA channel arbitration and between the core and DMA.

DMA Channel Registers

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

DMA Channel Allocation

Each channel has a set of parameter registers which are used to set up DMA transfers. [Table 2-28 on page 2-36](#) shows the DMA channel allocation and parameter register assignments for the ADSP-214xx processors.



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

Standard DMA Parameter Registers

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



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

Index registers. These registers, shown in [Table 2-2](#), provide an internal memory address that acts as a pointer to the next internal memory DMA read or write location. All internal index addresses are based on an internal memory offset of 0x80000.

Table 2-2. Index Registers

Register Name	Width (Bits)	Description
IISP0–7A	28	SPORTxA (supports external addresses)
IISP0–7B	28	SPORTxB (supports external addresses)
IISPI	19	SPI
IISPIB	19	SPIB
IDP_DMA_I0–7	19	IDPx
IDP_DMA_I0–7A	19	IDPx index A (ping pong)
IDP_DMA_I0–7B	19	IDPx index B (ping pong)
IIUART0RX	19	UART0 Receiver
IIUART0TX	19	UART0 Transmitter
IILB0–1	19	Link Port0–1
IIFIR	19	Accelerator FIR data input
CIFIR	19	Accelerator FIR coeff input
OIFIR	19	Accelerator FIR output
IIIIR	19	Accelerator IIR data input
CIIIR	19	Accelerator IIR coeff input

Table 2-2. Index Registers (Cont'd)

Register Name	Width (Bits)	Description
OIIIR	19	Accelerator IIR output
IIFTT	19	Accelerator FFT input
OIFFT	19	Accelerator FFT output
IIMTMW	19	MTM Write
IIMTMR	19	MTM Read
IIEP0–1	19	External Port0–1
EIEP0–1	28	External Port (external)

Modify registers. These registers, shown in [Table 2-3](#), provide the signed increment by which the DMA controller post-modifies the corresponding memory index register after the DMA read or write.

Table 2-3. Modify Registers

Register Name	Width (Bits)	Description
IMSP0–7A	16	SPORTA
IMSP0–7B	16	SPORTB
IMSPI	16	SPI
IMSPIB	16	SPIB
IDP_DMA_M0–7	6	IDP
IDP_DMA_M0–7A	6	IDP modify A (ping pong)
IDP_DMA_M0–7B	6	IDP modify B (ping pong)
IMLB0–1	16	Link Port
IMUART0RX	16	UART0 Receiver
IMUART0TX	16	UART0 Transmitter
IMFIR	16	Accelerator FIR data input
CMFIR	16	Accelerator FIR coeff input
OMFIR	16	Accelerator FIR output

DMA Channel Registers

Table 2-3. Modify Registers (Cont'd)

Register Name	Width (Bits)	Description
IMIIR	16	Accelerator IIR data input
CMIIR	16	Accelerator IIR coeff input
OMIIR	16	Accelerator IIR output
IMFFT	16	Accelerator FFT input
OMFFT	16	Accelerator FFT output
IMMTMW	16	MTM Write
IMMTMR	16	MTM Read
IMEP0–1	16	External Port
EMEP0–1	27	External Port (external)

Count registers. These registers, shown in [Table 2-4](#), indicate the number of words remaining to be transferred to or from memory on the corresponding DMA channel.

Table 2-4. Count Registers

Register Name	Width (Bits)	Description
ICSP0–7A	16	SPORTA
ICSP0–7B	16	SPORTB
ICSPI	16	SPI
ICSPIB	16	SPIB
IDP_DMA_C0–7	16	IDP
ICLB0–1	16	Link Port
CUART0RX	16	UART0 Receiver
CUART0TX	16	UART0 Transmitter
ICFIR	16	Accelerator FIR data input
CCFIR	16	Accelerator FIR coeff input
OCFIR	16	Accelerator FIR output

Table 2-4. Count Registers (Cont'd)

Register Name	Width (Bits)	Description
ICIIIR	16	Accelerator IIR data input
CCIIIR	16	Accelerator IIR coeff input
OCIIIR	16	Accelerator IIR output
ICFFT	16	Accelerator FFT input
OCFFT	16	Accelerator FFT output
ICMTMW	16	MTM Write
ICMTMR	16	MTM Read
ICEP0–1	16	External Port
ECEP0–1	16	External Port (external)

Chain pointer registers. These registers, shown in [Table 2-5](#), 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.



For information on transfer control blocks (TCBs), see “[DMA Chaining](#)” on page [2-32](#).

Table 2-5. Chain Pointer Registers

Register Name	Width (Bits)	Description
CPSPO-7A	28	SPORTA
CPSPO-7B	28	SPORTB
CPSPI	20	SPI
CPSPIB	20	SPIB
CPLB0–1	20	Link Port
CPUART0RX	20	UART0 Receiver
CPUART0TX	20	UART0 Transmitter

DMA Channel Registers

Table 2-5. Chain Pointer Registers (Cont'd)

Register Name	Width (Bits)	Description
CPFIR	20	Accelerator FIR
CPIIR	20	Accelerator IIR
CPIFFT	21	Accelerator FFT input
CPOFFT	20	Accelerator FFT output
CPEP0–1	21	External Port

Extended DMA Parameter Registers

This section describes the enhanced parameter registers used for Accelerator and External Port.

Base registers. These registers, shown in [Table 2-6](#), base registers indicate the start address of the circular buffer to be transferred to/from memory on the corresponding DMA channel.

Table 2-6. Base Registers

Register Name	Width (Bits)	Description
IBFIR	19	Accelerator FIR input
OBFIR	19	Accelerator FIR output
IBIIR	19	Accelerator IIR input
OBIIR	19	Accelerator IIR output
IBFFT	19	Accelerator FFT input
OBFFT	19	Accelerator FFT output
EPEP0–1	28	External Port (external base)

Length registers. These registers, shown in [Table 2-7](#), define the length of the circular buffer to be transferred to/from memory on the corresponding DMA channel.

Table 2-7. Length Registers

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

Miscellaneous External Port Parameter registers. These registers, shown in [Table 2-8](#), used for the delay line and scatter/gather DMA to read from tap list buffers, store counters and index pointers.

Table 2-8. Miscellaneous External Port Parameter Registers

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

MLB Parameter registers. Note that the MLB interface does not have specific parameter registers like the other peripherals. Instead it has many control registers whereby the base and offset addresses for a specific channel gets defined. [For more information, see Chapter 8, Media Local Bus.](#)

DMA Channel Registers

Data Buffers

The data buffers or FIFOs (shown in Table 2-9) are used by each DMA channel to store data during the priority arbitration time period. The buffers (depending on the peripheral) are accessed by both DMA and the core. Note that all transmit buffers are write-only-to-clear (WOC) and all receive buffers are read-only-to-clear (ROC).

Table 2-9. Data Buffers

Buffer Name	FIFO Depth	Description
TXSP0-7A	2	SPORTA Transmit
TXSP0-7B	2	SPORTB Transmit
RXSP0-7A	2	SPORTA Receive
RXSP0-7B	2	SPORTB Receive
TXSPI	2	SPI Transmit
TXSPIB	2	SPIB Transmit
RXSPI	2	SPI Receive
RXSPIB	2	SPIB Receive
RXSPI_SHADOW	2	SPI Receive Shadow (RO)
RXSPIB_SHADOW	2	SPIB Receive Shadow (RO)
SPI DMA	4	DMA only
SPIB DMA	4	DMA only
IDP_FIFO	8	IDP FIFO Receive
TXLB0-1	2	Link Port Transmit Buffer
TXLB0-1_IN_SHADOW	1	Link Port Transmit Shadow Buffer (RO)
TXLB0-1_OUT_SHADOW	1	Link Port Transmit Shadow Buffer (RO)
RXLB0-1	2	Link Port Receive Buffer
RXLB0-1_IN_SHADOW	1	Link Port Receive Shadow Buffer (RO)
RXLB0-1_OUT_SHADOW	1	Link Port Receive Shadow Buffer (RO)

Table 2-9. Data Buffers (Cont'd)

Buffer Name	FIFO Depth	Description
UARTRBR0	1	UART0 Receiver
UARTTHR0	1	UART0 Transmitter
Accelerator input	8	FFT DMA only
Accelerator output	8	FFT DMA only
MTM read/write	2	DMA only
DFEP0–1	6	DMA only
AMIRX	1	AMI Receive Packer
AMITX	1	AMI Transmit Packer
TXTWI8	1 (1 byte)	TWI Transmit
TXTWI16	1 (2 bytes)	TWI Transmit
RXTWI8	1 (1 byte)	TWI Receive
RXTWI16	1 (2 bytes)	TWI Receivet
MLB local buffer	124	MLB SRAM

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

Chain Pointer Registers

The chain pointer registers, described in [Table 2-10](#) (generic), [Table 2-11](#) (SPORTs) and [Table 2-12](#) (external port) are 20 bits wide. The lower 19 bits are the memory address field. Like other I/O processor address registers, the chain pointer register's value is offset to match the starting address of the processor's internal memory before it is used by the I/O processor. On the SHARC processor, this offset value is 0x80000.

DMA Channel Registers

Table 2-10. Chain Pointer Register (CPx)

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

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

Table 2-11. SPORT Chain Pointer Register (CPSPx)

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

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

Table 2-12. External Port Chain Pointer Register (EPCPx)

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

Table 2-13. FFT Input Chain Pointer Register (CPIFFT)

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

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

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



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

TCB Storage

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

Serial Port TCB

The serial ports support single and chained DMA. [Table 2-14](#) shows the required TCBs for chained DMA

Table 2-14. SPORT TCBs

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

SPI TCB

The serial peripheral interfaces supports both single and chained DMA. However, unlike the serial ports, programs cannot insert a TCB in an active chain. [Table 2-15](#) shows the required TCBs for chained DMA.

Table 2-15. SPI/SPIB TCBs

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

UART TCB

The UART interface supports both single and chained DMA. However, unlike the serial ports, programs cannot insert a TCB in an active chain. [Table 2-16](#) shows the required TCBs for chained DMA.

Table 2-16. UART0 TCBs

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

Link Port TCB

The link port interface supports both single and chained DMA. [Table 2-17](#) shows the required TCBs for chained DMA.

Table 2-17. Link Port TCBs

Address	Register
CP[18:0]	CPLPx Chain Pointer
CP[18:0] + 0x1	CLBx Internal Count
CP[18:0] + 0x2	IMLBx Internal Modifier
CP[18:0] + 0x3	IILBx Internal Index

FIR Accelerator TCB

The FIR accelerator DMA supports circular buffer chained DMA.

[Table 2-18](#) shows the required TCBs for chained DMA. The FIR accelerator does not support circular buffering for the coefficient buffer.

Table 2-18. FIR TCBs

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



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

IIR Accelerator TCB

The IIR accelerator supports circular buffer chained DMA. Table 2-19 shows the required TCBs for chained DMA.



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

Table 2-19. IIR TCBs

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

FFT Accelerator TCB

The FFT accelerator supports circular buffer chained DMA. [Table 2-20](#) and [Table 2-21](#) shows the required TCBs for chained DMA.

Table 2-20. FFT Input TCBs

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



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

Table 2-21. FFT Output TCBs

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

External Port TCB

The external port interface supports many different types of DMA, resulting in different lengths of TCBs. The TCB size varies from six locations (chained DMA) to 13 locations (delay line DMA). [Table 2-22](#) shows the required TCBs for chained DMA.

Table 2-22. External Port TCBs

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

The order the descriptors are fetched with circular buffering enabled is shown in [Table 2-23](#).

Table 2-23. External Port TCBs for Circular DMA

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

TCB Storage

For delay line DMA, TCB loading is split into two sequences to improve overall priority. The first TCB loads the write parameters (IIEP–ELEP) and the second loads the read parameters (RIEP–CPEP). This two stage loading is transparent to the application. The order the descriptors are fetched with circular buffering enabled is shown in [Table 2-24](#).

Table 2-24. External Port TCBs for Delay Line DMA

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

The order the descriptors are fetched for scatter/gather DMA with circular buffering enabled is shown in [Table 2-25](#) and [Table 2-26](#).

Table 2-25. External Port TCBs for Scatter/Gather DMA

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

Table 2-26. External Port TCBs for Circular Scatter/Gather DMA

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

Clocking

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

Functional Description

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

Automated Data Transfer

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

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

DMA Transfer Types

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

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

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

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



The IDP port does not support DMA chaining.

Ping-pong DMA (IDP). In ping-pong DMA, the parameters have two memory index values (index A and index B), one count value and one modifier value. The DMA starts the transfer with the memory indexed by

Functional Description

A. When the transfer is completed as per the value in the count register, the DMA restarts with the memory location indexed by B. The DMA restarts with index A after the transfer to memory with index B is completed as per the count value. This repeats until the DMA is stopped by resetting the DMA enable bit.

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

DMA Direction

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

Internal to External Memory

DMA transfers between internal memory and external memory devices use the processor's external port. For these types of transfers, the application code provides the DMA controller with the internal memory buffer size, address, and address modifier, as well as the external memory buffer size, address, address modifier, and the direction of transfer. After setup, the DMA transfers begin when the program enables the channel and continues until the I/O processor transfers the entire buffer to processor memory. [Table 2-28 on page 2-36](#) shows the parameter registers for each DMA channel.

Peripheral to Internal Memory

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

The direction (receive or transmit) of the peripheral determines the direction of data transfer. When the port receives data, the I/O processor automatically transfers the data to internal memory. When the port needs to transmit a word, the I/O processor automatically fetches the data from internal memory. [Figure 2-1 on page 2-26](#) shows more detail on DMA channel data paths.

Peripheral to External Memory (SPORTs)

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

Internal Memory to Internal Memory

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

DMA Controller Addressing

[Figure 2-1](#) shows a block diagram of the I/O processor's address generator (DMA controller). [“Standard DMA Parameter Registers” on page 2-4](#) lists the parameter registers for each DMA channel. The parameter registers are uninitialized following a processor reset.

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

Functional Description

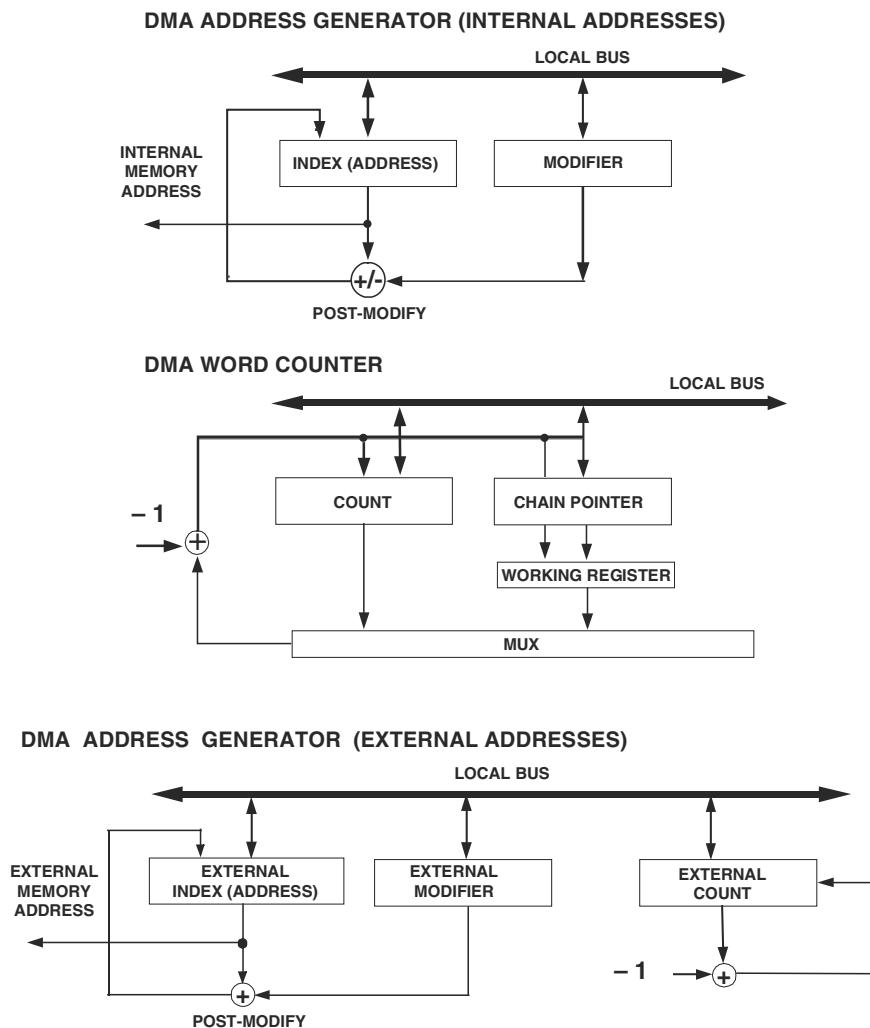


Figure 2-1. DMA Address Generator

Internal Index Register Addressing

All addresses in the index registers are offset by a value matching the processor's first internal normal word addressed RAM location, before the I/O processor uses the addresses. For the ADSP-214xx processors, this offset value is 0x0008 0000.

The following rules for data transfers must be followed.

- DMA index addresses must always be normal word space (32-bit).
- The I/O processor can transfer short word data (16-bit or 8-bit) using the packing capability of the peripherals (serial port or SPI). The data are packed in the peripheral's shift register to form 32-bit words for the internal transfers over the IOD0 and IOD1 buses.

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

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



If a program loads the count register with zero, the I/O processor does not disable DMA transfers on that channel. The I/O processor interprets the zero as a request for 2^{16} transfers. This count

Functional Description

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.

External Index Register Addressing

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

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

DMA Channel Status

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

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



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

The peripheral's DMA controller tracks status information of the channels in each of the peripheral registers (for example SPMCTL x , SPIDMAC x , DAI_STAT, DMAC x , and MTMCTL).

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

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

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

DMA Start and Stop Conditions

The difference between single DMA and chained DMA is based on the auto-linkage process where the DMA's attributes are stored in internal memory and automatically loaded by the IOP if requested.

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 non zero value. In this case, TCB chain loading of the channel parameter registers occurs first.
- Chaining is enabled, the chain pointer register address field is non-zero, and the current DMA sequence finishes. Again, TCB chain loading occurs.

Operating Modes

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 (SPORT only) and the DMA sequence continues.

Once a program starts a DMA process, the process is influenced by two external controls—DMA channel priority and DMA chaining.

Operating Modes

This section provides information on IOP operating modes.

The SHARC processor contains two independent 32-bit DMA buses ([Figure 2-2](#)). The IOD0 bus is used for the peripherals to the internal memory and the IOD1 bus is used for external-to-internal memory transfers.

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

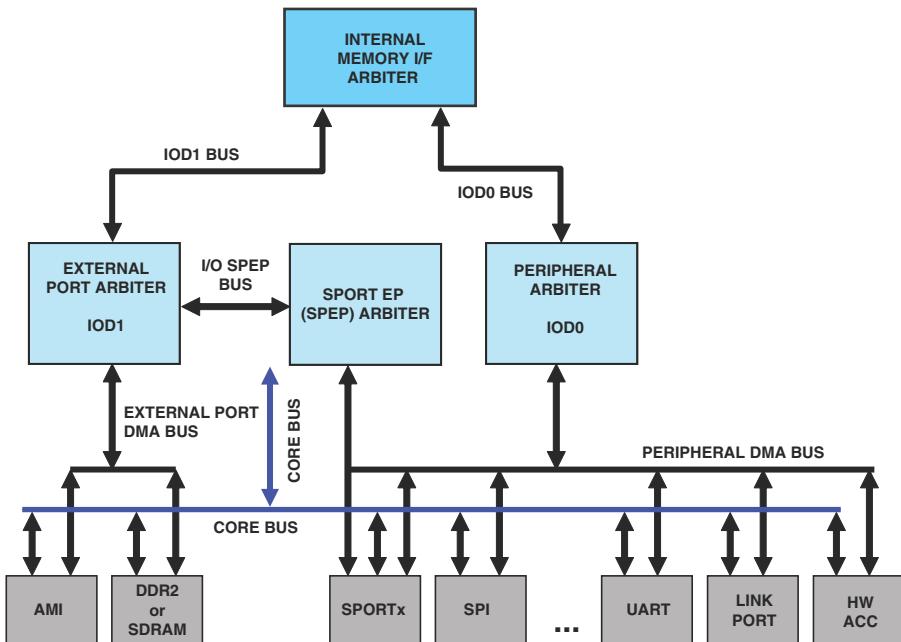


Figure 2-2. I/O Processor Bus Structure



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

Each I/O port has one or more DMA channels, and each channel has a single request and a single grant. When a particular channel needs to read or write data to internal memory, the channel asserts an internal DMA request. The I/O processor prioritizes the request with all other valid DMA requests. When a channel becomes the highest priority requester, the I/O processor asserts the channel's internal DMA grant. In the next clock cycle, the DMA transfer starts. [Table 2-28 on page 2-36](#) shows the paths for internal DMA requests within the I/O processor.

DMA Chaining

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

-  If chaining is enabled on a DMA channel, programs should not use polling to determine channel status as this gives inaccurate information where the DMA appears inactive if it is sampled while the next TCB is loading.

TCB Memory Storage

The location of the DMA parameters for the next sequence comes from the chain pointer register that points to the next set of DMA parameters stored in the processor's internal memory. In chained DMA operations, the processor automatically initializes and then begins another DMA transfer when the current DMA transfer is complete. Each new set of parameters is stored in a user-initialized memory buffer or TCB for a chosen peripheral. [Table 2-27](#) provides a brief description of the TCBs.

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

Table 2-27. Principal TCB Allocation for a Serial Peripheral

Address	Register	Description
CPx	Chain pointer register	Chain pointer for DMA chaining
CPx + 0x1 (ICx)	Internal count register	Length of internal buffer

Table 2-27. Principal TCB Allocation for a Serial Peripheral (Cont'd)

Address	Register	Description
CPx + 0x2 (IMx)	Internal modify register	Stride for internal buffer
CPx + 0x3 (IIx)	Internal index register	Internal memory buffer

Chain Assignment

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

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

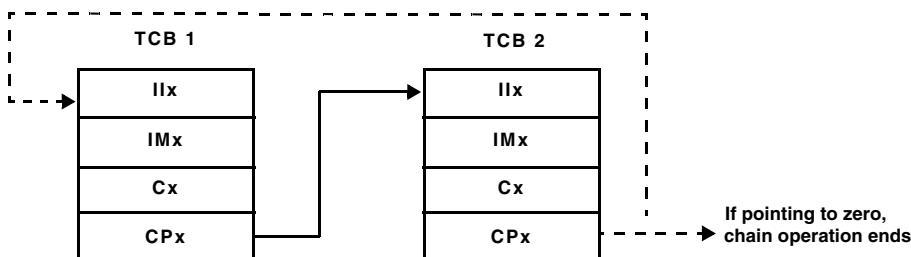


Figure 2-3. Chaining in the SPI and Serial Ports

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



Clear the chain pointer register before chaining is enabled.

Operating Modes

Listing 2-1. Chain Assignment

```
R0=0;  
dm(CPx)=R0;           /* clear CPx register */  
  
/* init DMA control registers */  
  
R2=(TCB1+3) & 0xFFFF;    /* load IIx address of next TCB  
                         and mask address */  
R2=bset R2 by 19;        /* set PCI bit */  
dm(TCB2)=R2;            /* write address to CPx location of  
                         current TCB */  
R2=(TCB2+3) & 0xFFFF;    /* load IIx address of next TCB and  
                         mask address*/  
R2=bclr R2 by 19;        /* clear PCI bit */  
dm(TCB1)=R2;            /* write address to CPx location of  
                         current TCB */  
  
dm(CPx)=R2;             /* write IIx address of TCB1 to CPx  
                         register to start chaining*/
```



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

Starting Chain Loading

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

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

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



When starting chain loading, note that the SPI port is an exception to the above. To execute the first DMA in a chain for this peripheral, the DMA parameter registers also need to be explicitly programmed. [For more information, see “DMA Transfers” on page 15-21.](#)

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

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 = 0x0) until some event occurs that loads the chain pointer register with a non zero value. Writing all zeros to the address field of the chain pointer register also disables chaining.

TCB Chain Loading Priority

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



A channel that is in the process of chain loading cannot be interrupted by any other request (TCB, DMA channel). The chain loading sequence is atomic and the I/O bus is locked until all the DMA parameter registers are loaded. For a list of DMA channels in priority order, see [Table 2-28](#).

Operating Modes

Chain Insert Mode (SPORTs Only)

It is possible to insert a single SPORT DMA operation or another DMA chain within an active SPORT DMA chain. Programs may need to perform insertion when a high priority DMA requires service and cannot wait for the current chain to finish. This is supported only for SPORT DMA channels. [For more information, see Chapter 10, Serial Ports.](#)

Fixed DMA Channel Arbitration

The shaded region in [Table 2-28](#) (DMA channels 65 and 66) illustrates that the priority shown is only valid if the IOD1 bus (external port DMA channels) has a memory address block conflict with the IOD0 peripherals bus. Otherwise, the IOD1 bus operates fully independently. Also note the external port DMA channel changes priority, depending on the external index addresses. If an external index address is assigned to an internal index address, then DMA channel priority will change.

Table 2-28. DMA Channel 0–66 Priorities

DMA Channel Number	Peripheral Group	Control/Status Registers	Parameter Registers	Data Buffer	Description
IOD0 Peripheral Bus					
0 (Highest Priority)	A	SPCTL1, SPMCTL1	IISP1A, IMSP1A, CSP1A, CPSP1A	RXSP1A or TXSP1A	Serial Port 1A Data
1			IISP1B, IMSP1B, CSP1B, CPSP1B	RXSP1B or TXSP1B	Serial Port 1B Data
2		SPCTL0, SPMCTL0	IISP0A, IMSP0A, CSP0A, CPSP0A	RXSP0A or TXSP0A	Serial Port 0A Data
3			IISP0B, IMSP0B, CSP0B, CPSP0B	RXSP0B or TXSP0B	Serial Port 0B Data

Table 2-28. DMA Channel 0–66 Priorities (Cont'd)

DMA Channel Number	Peripheral Group	Control/Status Registers	Parameter Registers	Data Buffer	Description
4	B	SPCTL3, SPMCTL3	IISP3A, IMSP3A, CSP3A, CPSP3A	RXSP3A or TXSP3A	Serial Port 3A Data
5			IISP3B, IMSP3B, CSP3B, CPSP3B	RXSP3B or TXSP3B	Serial Port 3B Data
6		SPCTL2, SPMCTL2	IISP2A, IMSP2A, CSP2A, CPSP2A	RXSP2A or TXSP2A	Serial Port 2A Data
7			IISP2B, IMSP2B, CSP2B, CPSP2B	RXSP2B or TXSP2B	Serial Port 2B Data
8	C	SPCTL5, SPMCTL5	IISP5A, IMSP5A, CSP5A, CPSP5A	RXSP5A or TXSP5A	Serial Port 5A Data
9			IISP5B, IMSP5B, CSP5B, CPSP5B	RXSP5B or TXSP5B	Serial Port 5B Data
10		SPCTL4, SPMCTL4	IISP4A, IMSP4A, CSP4A, CPSP4A	RXSP4A or TXSP4A	Serial Port 4A Data
11			IISP4B, IMSP4B, CSP4B, CPSP4B	RXSP4B or TXSP4B	Serial Port 4B Data

Operating Modes

Table 2-28. DMA Channel 0–66 Priorities (Cont'd)

DMA Channel Number	Peripheral Group	Control/Status Registers	Parameter Registers	Data Buffer	Description
12	D	IDP_CTL, IDP_CTL1, IDP_CTL2, IDP_PP_CTL, DAI_STAT	IDP_DMA_I0, IDP_DMA_M0, IDP_DMA_C0, IDP_DMA_I0A, IDP_DMA_I0B, IDP_DMA_PC0	IDP_FIFO	DAI IDP or PDAP (only channel 0 supports both)
13			IDP_DMA_I1, IDP_DMA_M1, IDP_DMA_C1, IDP_DMA_I1A, IDP_DMA_I1B, IDP_DMA_PC1		Serial Input DAI IDP Channel 1
14			IDP_DMA_I2, IDP_DMA_M2, IDP_DMA_C2, IDP_DMA_I2A, IDP_DMA_I2B, IDP_DMA_PC2		Serial Input DAI IDP Channel 2
15			IDP_DMA_I3, IDP_DMA_M3, IDP_DMA_C3, IDP_DMA_I3A, IDP_DMA_I3B, IDP_DMA_PC3		Serial Input DAI IDP Channel 3
16			IDP_DMA_I4, IDP_DMA_M4, IDP_DMA_C4, IDP_DMA_I4A, IDP_DMA_I4B, IDP_DMA_PC4		Serial Input DAI IDP Channel 4

Table 2-28. DMA Channel 0–66 Priorities (Cont'd)

DMA Channel Number	Peripheral Group	Control/Status Registers	Parameter Registers	Data Buffer	Description
17	D	IDP_CTL, IDP_CTL1, IDP_CTL2, IDP_PP_CTL, DAI_STAT	IDP_DMA_I5, IDP_DMA_M5, IDP_DMA_C5, IDP_DMA_I5A, IDP_DMA_I5B, IDP_DMA_PC5	IDP_FIFO	Serial Input DAI IDP Channel 5
18			IDP_DMA_I6, IDP_DMA_M6, IDP_DMA_C6, IDP_DMA_I6A, IDP_DMA_I6B, IDP_DMA_PC6		Serial Input DAI IDP Channel 6
19			IDP_DMA_I7, IDP_DMA_M7, IDP_DMA_C7, IDP_DMA_I7A, IDP_DMA_I7B, IDP_DMA_PC7		Serial Input DAI IDP Channel 7
20	E	SPICTL, SPIDMAC, SPIBAUD SPISTAT	IISPI, IMSPI, CSPI, CPSPI	RXSPI or TXSPI and DMA Buffer	SPI Data
21–51	F	MLB_xCR		Local SRAM Buffer	MLB Data
52	G	SPICTLB, SPIDMACB, SPIBAUDB, SPISTATB	IISPIB, IMSPIB, CSPIB, CPSPIB	RXSPIB or TXSPIB and DMA Buffer	SPI B Data
53	H	MTMCTL (or DTCP)	IIMTMW, IMMTMW, CMTMW	MTM FIFO	Memory-to- memory write data
54	I		IIMTMR, IMMTMR, CMTMR	MTM FIFO	Memory-to- memory read data

Operating Modes

Table 2-28. DMA Channel 0–66 Priorities (Cont'd)

DMA Channel Number	Peripheral Group	Control/Status Registers	Parameter Registers	Data Buffer	Description
55	J	UART0RXCTL, UART0RXSTAT	IIUART0RX, IMUART0RX, CUART0RX, CPUART0RX,	UARTRBR0 Buffer	UART0 Receive Buffer Register
56		UART0TXCTL, UART0TXSTAT	IIUART0TX, IMUART0TX, CUART0TX, CPUART0TX,	UARTTHR0 Buffer	UART0 Transmit Holding Register
57	K	LCTL0, LSTAT0	IILB0, IMLB0, ICLB0, CPLB0	TXLB0 or RXLB0	Link Port 0 Data (ADSP-2146x only)
58		LCTL1, LSTAT1	IILB1, IMLB1, ICLB1, CPLB1	TXLB0 or RXLB0	Link Port 1 Data (ADSP-2146x only)
59	L	SPCTL7, SPMCTL7	IISP7A, IMSP7A, CSP7A, CPSP7A	RXSP7A or TXSP7A	Serial Port 7A Data
60			IISP7B, IMSP7B, CSP7B, CPSP7B	RXSP7B or TXSP7B	Serial Port 7B Data
61		SPCTL6, SPMCTL6	IISP6A, IMSP6A, CSP6A, CPSP6A	RXSP6A or TXSP6A	Serial Port 6A Data
62			IISP6B, IMSP6B, CSP6B, CPSP6B	RXSP6B or TXSP6B	Serial Port 6B Data

Table 2-28. DMA Channel 0–66 Priorities (Cont'd)

DMA Channel Number	Peripheral Group	Control/Status Registers	Parameter Registers	Data Buffer	Description
63	M	PMCTL1, FIRCTL1, FIRCTL2, FIRMACSTAT, FIRDMASTAT	IIFIR, IMFIR, ICFIR, IBFIR, CIFIR, CMFIR, CLFIR, CPFIR	Accelerator Input Buffers and FIFO	FIR, IIR, FFT Accelerator Input Data
		PMCTL1, IIRCTL1, IIRCTL2, IIRMACSTAT, IIRDMASTAT	IIIIR, IMIIR, ICIIR, IBIIR, CIIR, CMIIR, CLIIR, CPIIR		
		PMCTL1 FFTCTL1, FFTCTL2, FTTMACSTAT, FFTDMASTAT	IIFT, IMFFT, ICFFT, IBFFT, CIFT, CMFFT, CLFFT, CPIFFT,		
64	N	PMCTL1, FIRCTL1, FIRCTL2, FIRMACSTAT, FIRDMASTAT	OIFIR, OMFIR, OCFIR, OBFIR, COFIR, CMFIR, CLFIR, CPFIR	Accelerator Out- put Buffers and FIFO	FIR, IIR, FFT Accelerator Out- put Data
		PMCTL1, IIRCTL1, IIRCTL2, IIRMACSTAT, IIRDMASTAT	IIIIR, IMIIR, ICIIR, IBIIR, CIIR, CMIIR, CLIIR, CPIIR		
		PMCTL1, FFTCTL1, FFTCTL2, FTTMACSTAT, FFTDMASTAT	OIFFT, OMFFT, OCFFT, OBFFT, COFFT, CMFFT, CLFFT, CPOFFT,		

Operating Modes

Table 2-28. DMA Channel 0–66 Priorities (Cont'd)

DMA Channel Number	Peripheral Group	Control/Status Registers	Parameter Registers	Data Buffer	Description
IOD1 External Port Bus					
65	Q	DMAC0	IIEP0, IMEP0, ICEP0, EIEP0, EMEP0 ELEP0, EBEP0 RIEP0, RCEP0 RMEP0, EPTC0, EPTP0, EPCP0	DFEP0 and AMIRX AMITX (AMI only)	External Port Memory DMA 0
66	R	DMAC1	IIEP1, IMEP1, ICEP1, EIEP1, EMEP1 ELEP1, EBEP1 RIEP1, RCEP1 RMEP1, EPTC1, EPTP1, EPCP1	DFEP1 and AMIRX AMITX (AMI only)	External Port Memory DMA 1. Note if the DMAC0 channel runs int-int mem- ory and DMAC1 channel int-ext memory, then DMAC1 has higher priority.

Peripheral DMA Bus

DMA-capable peripherals execute DMA data transfers to and from internal memory over the IOD0 bus. When more than one of these peripherals requests access to the IOD0 bus in a clock cycle, the bus arbiter, which is attached to the IOD0 bus, determines which master should have access to the bus and grants the bus to that master.

IOP channel arbitration can be set to use either a fixed or rotating algorithm by setting or clearing DCPR bit in the `SYSCTL` register as follows.

- (=0) fixed arbitration (default)
- (=1) rotating arbitration

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

External Port DMA Bus

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

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

- (=10) fixed arbitration channel 0 high
- (=11) rotating arbitration (default)

Note the independency is only broken if there is an internal memory block conflict. In this case, if both rotating bits are set, the peripheral DMA channels always have the highest priority and the DMAPR bit allows the change in priority among the two external port DMA channels.

SPORT/External Port DMA Bus

The data connection between the SPORT and the external port is performed over the SPEP (SPORT/External Port) DMA bus. Note that it is possible to run two independent SPORT DMA channels in parallel: the first over the SPORT to the external port and the second from any other SPORT to the internal memory.

Operating Modes

External port/SPORT channel arbitration can be set to use either a fixed or rotating algorithm by setting the EPBR bits in the EPCTL register as follows.

- (00) = Priority order from highest to lowest is SPORT, external port DMA, core.
- (01) = Priority order from highest to lowest is external port DMA, SPORT, core.
- (10) = Highest priority is core. SPORT and external port DMA are in rotating priority.

Rotating DMA Channel Arbitration

DMA channel arbitration is the method that the arbiter uses to determine how groups rotate priority with other channels. The default DMA channel priority is fixed prioritization by DMA channel group.

Rotating Priority by Group

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 by DMA channel. Peripheral groups are shown in [Table 2-28](#).

Initially, group A has the highest priority and group I 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 2-28](#)). For example, of the SPORT pair SP01 (which

is in group A), SP1 has the highest priority. Programs can change DMA arbitration modes between fixed and rotate on the fly which incurs an effect latency of 2 PCLK cycles.

Interrupts

The primary type of DMA communication is interrupt driven I/O where the core continues to execute instructions while DMA executes in the background. This allows high levels of parallelism achieving over all better system performance. Because the interrupt vector directs the core to respond to specific transactions very efficiently, programs do not need to poll status bits.

During interrupt-driven DMA, programs use the interrupt mask bits in the IMASK, LIRPTL, DPI_IMASK, DAI_IMASK_x registers to selectively mask DMA channel interrupts that the I/O processor latches into the IRPTL, LIRPTL, DPI_IMASK, DAI_IMASK_x registers. A channel interrupt mask in the IMASK, LIRPTL, DPI_IMASK, DAI_IMASK_x registers determines whether a latched interrupt is serviced or not. When an interrupt is masked, it is latched but not serviced.

Sources

The following sections describe the two sources of interrupts.

Unchained DMA Interrupts

When an unchained (single block) DMA process reaches completion (the DMA 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, DPI_IMASK, or DAI_IMASK_x registers.

Interrupts

Chained DMA Interrupts

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

1. If $\text{PCI} = 1$, (bit 19 of the chain pointer register is the program controlled interrupts, or PCI bit) an interrupt occurs for each DMA in the chain.
2. If $\text{PCI} = 0$, an interrupt occurs at the end of a completed chain. For more information on DMA chaining, see “[Functional Description](#)” on page 2-22.

Figure 2-4 shows the PCI timing during TCB loading. After the DMA count for the last word of frame N becomes zero, the PCI interrupt is latched. At the same time the DMA reloads the TCB for that specific channel (assuming no higher priority DMA requests). Finally the DMA channel resumes operation for frame N-1.

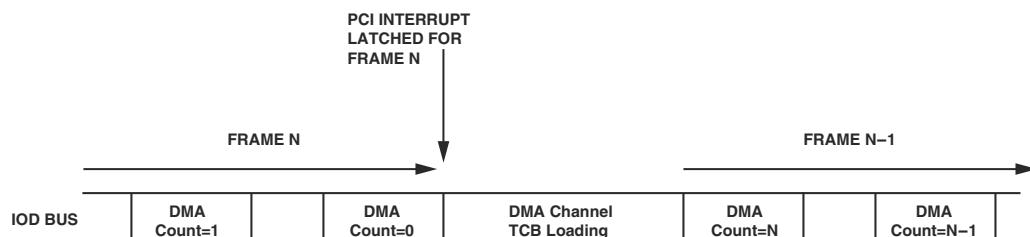


Figure 2-4. DMA Chaining



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.

Transfer Completion Types

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

Internal Transfer Completion

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



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

Access Completion

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

This mode is supported by the SPORT, SPI, link ports, and external port.

Core Single Word Transfer Interrupts

When a DMA channel's buffer is not being used for a DMA process, the core can generate an interrupt on single word transfers (writes or reads) of the buffer of the respective peripheral. This interrupt service differs slightly for each peripheral. In this case, the peripheral's buffer 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

Interrupts

interrupt-driven I/O under control of the processor core. Refer to the specific peripheral chapter for more information.

Interrupt Versus Channel Priorities

At their default setting shown in [Table 2-29](#), the DMA interrupt priorities do not match the DMA channel priorities. However, if both priorities schemes should match, the DMA interrupt priorities can be re-assigned by dedicated settings of the `PICRx` registers.

Table 2-29. Default Channel vs. Interrupt Priorities

Programmable Interrupt	Default Interrupt Priority	Priorities	DMA Channel Priority
POI	DAIHI	Highest	SPORT5–0, 12 channels
P1I	SPII		IDP7–0, 7 channels
P3I	SP1I		
P4I	SP3I		
P5I	SP5I		SPI – 1 channel
P6I	SP0I		
P7I	SP2I		MLB – 31 channels
P8I	SP4I		
P9I	EP0I		SPI B – 1 channel
P11I	SP7I		
P12I	DAILI		MTM (WR/RD) – 2 channels
P13I	EP1I		
P14I	DPII		UART0(Tx/Rx) – 2 channels
P15I	MTMI		
P16I	SP6I		SPORT7–6, 4 channels
P17I	No default		
P18I	SPIBI		Accelerator (In/Out) 2 channels EPDMA1–0, 2 channels

Debug Features

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

Emulation Considerations

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

Effect Latency

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

Write Effect Latency

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

IOP Effect Latency

Table 2-30 lists the time required to load a specific TCB from the internal memory into the DMA controller. During this time, both buses (for a peripheral DMA, the IOD0 bus and for external port DMA the IOD1 bus) are locked and cannot be interrupted.

Programming Model

IOP Throughput

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

- Internal memory writes with $f_{PCLK} \times 32\text{-bit}$
- Internal memory reads with $f_{PCLK}/2 \times 32\text{-bit}$

Table 2-30. I/O Processor TCB Chain Loading Access

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

1 If the TCB for a SPORT is located in external memory, additional access cycles are required for External Port arbitration and AMI or DDR2 cycles.

2 For throughput performance add 6 core cycles.

Programming Model

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

General Procedure for Configuring DMA

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

1. Clear all relevant registers (DMA/peripheral control, chain pointer).
2. Determine interaction method (enable `IRQEN/IMASK` setting or status polling).
3. Define the DMA channels interrupt priority (`PICR` registers).
4. Determine the DMA channel priority (fixed or rotating).
5. Determine the DMA address region for source and destination (index, modifier, count).
6. Determine the DMA transfer type (standard, chained, circular).
7. Set the DMA enabled bit/write index address of first TCB to the chain pointer register.

Programming Model

3 EXTERNAL PORT

The external memory interface provides a glueless interface to external memories. The asynchronous memory interface and the SDRAM/DDR2 memory that interfaces to the external port is clocked by the SDRAM or DDR2 clock. The interface specifications are shown in [Table 3-1](#).

Table 3-1. External Port Specifications

Feature	AMI	SDRAM Interface	DDR2 Interface
Connectivity			
Multiplexed Pinout	Yes (External Port)	Yes (External Port)	No
SRU DAI Required	No	No	No
SRU DAI Default Routing	N/A	N/A	N/A
SRU2 DPI Required	No	No	No
SRU2 DPI Default Routing	N/A	N/A	N/A
Interrupt Control	Yes	Yes	Yes
Protocol			
Master Capable	Yes	Yes	Yes
Slave Capable	No	No	No
Transmission Simplex	Yes	Yes	Yes
Transmission Half-Duplex	Yes	Yes	Yes
Transmission Full-Duplex	No	No	No
Access Type			
Data Buffer	Yes	Yes	Yes
Core Data Access	Yes	Yes	Yes

Features

Table 3-1. External Port Specifications (Cont'd)

Feature	AMI	SDRAM Interface	DDR2 Interface
DMA Data Access	Yes	Yes	Yes
DMA Channels	2	2	2
DMA Chaining	Yes	Yes	Yes
Boot Capable	Yes	No	No
Clock Operation	SDCLK or DDR2CLK	SDCLK	DDR2CLK

Features

The external port has the following features.

- Supports access to the external memory by core and DMA accesses. The external memory address space is divided in to four banks. Any bank can be programmed as either asynchronous or synchronous memory.
- An asynchronous memory interface which communicates with SRAM, FLASH, and other devices that meet the standard asynchronous SRAM access protocol.
- A SDRAM controller that supports a glue-less interface with any of the standard SDRAMs.
- A DDR2 controller that supports a glue-less interface with any of the standard DDR2.

- Arbitration Logic to coordinate core, SPORT DMA, external port DMA, transfers between internal and external memory over the external port.
- External port supports various ratios of core to external port clock determined by programming bits in the power management control registers ([PMCTL](#)). For more information, see “[ADSP-2147x/ADSP-2148x Power Management Registers](#)” on page A-12.

Pin Descriptions

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

Pin Multiplexing

The address, data and memory select pins are multiplexed for the AMI and the SDRAM controller but not for the DDR2 controller. For more information on multiplexing schemes refer to “[Pin Multiplexing](#)” on page 23-28.

Register Overview

This section provides brief descriptions of the major registers. For complete register information, see [Appendix A, Registers Reference](#).

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

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

Register Overview

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

Power Management Control Register (PMCTL). Controls the SDCLK to core clock ratio related to the AMI or SDRAM timing.

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

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

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

DDR2CTL0 register contains the bits that control the DDR2 size, enables mode register and allows forcing of specific DDR2 commands.

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

DDR2CTL2 register includes the programmable parameters associated to the burst type, burst length and CAS latency.

DDR2CTL5–3 registers include the programmable parameters associated with the DDR2 extended mode registers 1 through 3.

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

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

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

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

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

Clocking AMI/SDRAM

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

The AMI/SDRAM controller is capable of running at up to 166 MHz and can run at various frequencies, depending on the programmed SDRAM clock (SDCLK) to core clock (CCLK) ratios. The various possible AMI/SDRAM clock to core clock frequency ratios are shown in [Table 3-2](#).

For more information on clock settings refer to TBD IN PWR MGMT. For information processor instruction rates, see the appropriate processor data sheets.

Clocking AMI/DDR2

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

Table 3-2. External Port Clock Frequencies

CCLK:SDCLK Clock Ratio	CCLK = 400 MHz	CCLK = 333 MHz	CCLK = 266 MHz	CCLK = 200 MHz
1:2.0	N/A	166	133	100
1:2.5	160	133	106	80
1:3.0	133	111	88	67
1:3.5	114	95	76	57
1:4.0	100	83	66	25

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

Clocking AMI/DDR2

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

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

External Port Arbitration

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

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

Functional Description

The external port has four ports for communication:

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

[Figure 3-1](#) shows a diagram of the external port for the ADSP-2147x and ADSP-2148x processors (containing a SDRAM interface).

Functional Description

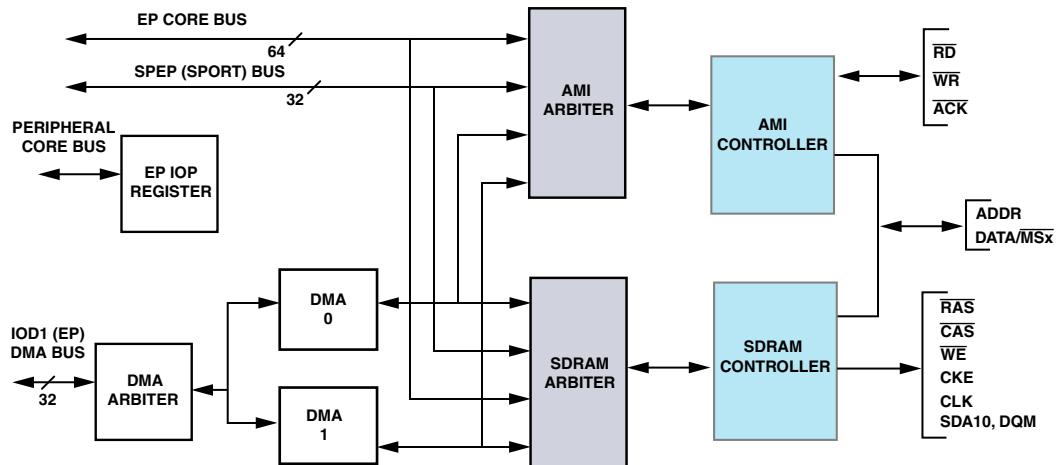


Figure 3-1. External Port Functional Block Diagram (SDRAM)

Figure 3-2 shows a diagram of the external port for the ADSP-2146x processor (containing a DDR2 interface).

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

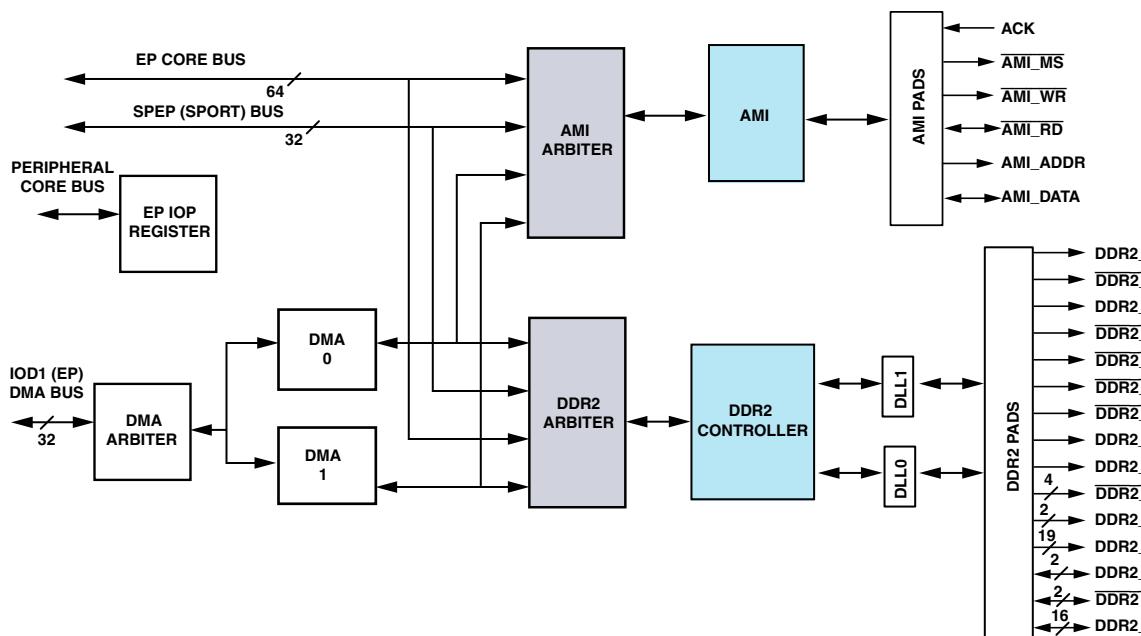


Figure 3-2. External Port Functional Block Diagram (DDR2)

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

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

In the EPCTL register, the EBPR and DMAPR bits define the priorities. All the bits of EPCTL register can be changed only when the external port is idle

Asynchronous Memory Interface

(when all DMA engines are idle and no core or SPORT access to external memory are pending).

Operating Mode

The following operation mode applies to the external port arbiter.

Arbitration Freezing

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

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

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



Channel freezing has no effect on write accesses.

Asynchronous Memory Interface

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

Features

The AMI has the following features and capabilities.

- User defined combinations of programmable wait states
- External hardware acknowledge signals
- Data packing support for 8 and 16 bits (ADSP-2147x and ADSP-2148x)
- External instruction fetch from 8 and 16 bits
- Both the processor core and the I/O processor have access to external memory using the AMI.
- Support a glueless interface with any of the standard SRAMs.
- Bank 0 can accommodate up to 6.2M words, and banks 1, 2, and 3 can accommodate up to 8M words each (ADSP-2147x and ADSP-2148x)
- Bank 0 can accommodate up to 2M words, and banks 1, 2, and 3 can accommodate up to 4M words each (ADSP-2146x)

Functional Description

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

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

Asynchronous Memory Interface

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

Asynchronous Reads

[Figure 3-3](#) shows an asynchronous read bus cycle. Asynchronous read bus cycles proceed as follows.

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

Asynchronous Writes

[Figure 3-4](#) shows an asynchronous write bus cycle. Asynchronous write bus cycles proceed as follows.

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

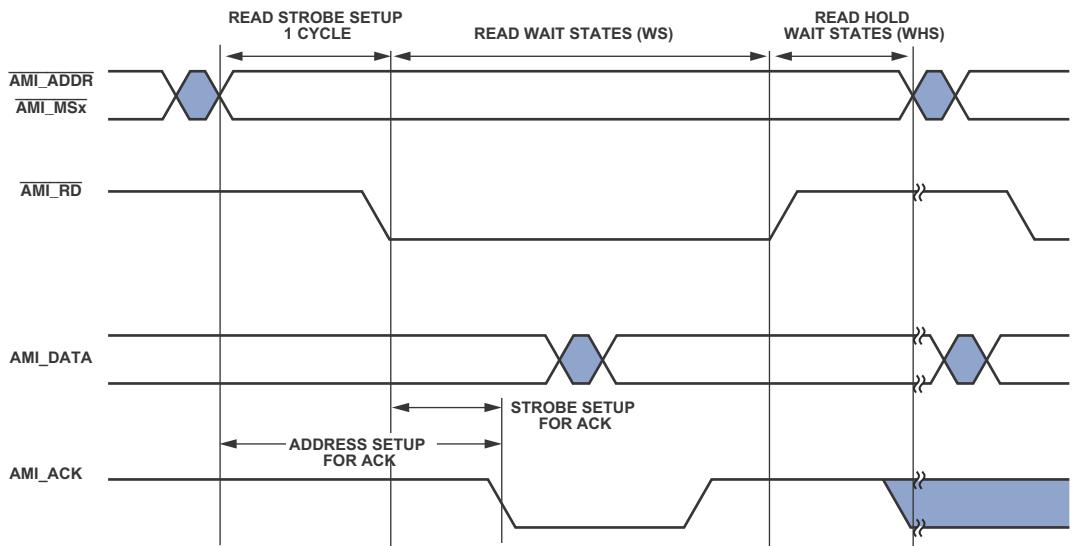


Figure 3-3. AMI Asynchronous Reads

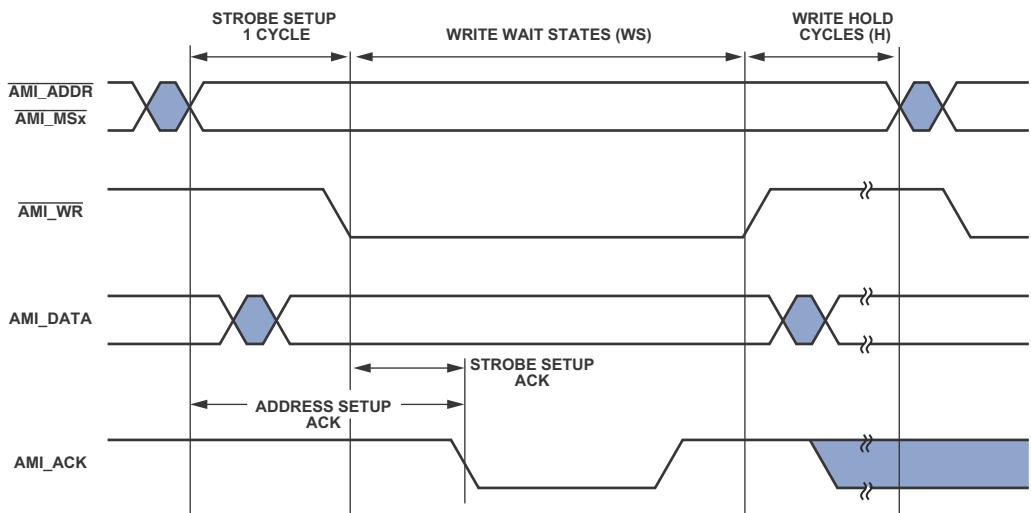


Figure 3-4. AMI Asynchronous Writes

Asynchronous Memory Interface

Parameter Timing

This section describes the programmable timing parameter for the AMI. The AMI controller allows to program access timing parameters (wait states for idle or hold cycles) with the effect being flexible and efficient whether initiation is from the core or from DMA, and the sequence of transactions (read followed by read, read followed by write, and so on).

Idle Cycles

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

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

Address Mapping

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

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

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

Table 3-3. AMI Address Memory Map

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

Operating Modes

The AMI operating modes are described in the following sections.

Data Packing

The combination of the (PKDIS) and (MSWF) bits allow combinations of packing for 8/16 to 32 bits. These modes are summarized in [Table 3-4](#).

External Access Extension

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

Asynchronous Memory Interface

Table 3-4. Data Packing Bit Settings (PKDIS)

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

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

Predictive Reads

The AMI controller allows two types of read access:

- predictive reads (default)
- non predictive reads

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

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

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



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

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

The SHARC processors support a glueless interface with any of the standard SDRAMs. The following sections provide detail about this interface.

Features

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 interface has the following additional features.

- I/O width 16-bits, I/O supply 3.3 V
- Types of 32, 64, 128, 256, and 512M bit with I/O of x4, x8, and x16
- Page sizes of 128, 256, 512, 1k, 2k words
- Variable memory address map (bank or page interleaving)

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

- Supports up to 254M words of SDRAM memory
- No-burst mode ($BL = 1$) with sequential burst type
- Open page policy—any open page is closed only if a new access in another page of the same bank occurs
- Supports multibank operation within the SDRAM
- Uses a programmable refresh counter to coordinate between varying clock frequencies and the SDRAM’s required refresh rate
- Provides multiple timing options to support additional buffers between the processor and SDRAM
- Allows independent auto-refresh while the asynchronous memory interface (AMI) has control of the external port
- Supports self-refresh mode for power savings
- Predictive data accesses for higher read data throughput (read optimization)
- Supports external instruction fetch in bank 0 for ISA and VISA operation
- Supports 64-bit SIMD mode by the core
- Supports dual data instruction type 1

Functional Description

The SDRAM control signals (\overline{MSx} , \overline{SDCKE} , \overline{SDRAS} , \overline{SDCAS} , \overline{SDWE}) define various operation modes to the SDRAM. [Table 3-5](#) provides a reference to these commands and the pin state for each one.

The configuration is programmed in the \overline{SDCTL} register. The SDRAM controller can hold off the processor core or DMA controller with an

internally connected acknowledge signal, as controlled by refresh, or page miss latency overhead.

A programmable refresh counter is provided which generates background auto-refresh cycles at the required refresh rate based on the clock frequency used. The refresh counter period is specified with the RDIV field in the SDRAM refresh rate control register (“[Refresh Rate Control Register \(SDRRC\)](#)” on page A-58).

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

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

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

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

The 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 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.

SDRAM 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 3-5 on page 3-25](#).

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

Load Mode Register

This command is initializes SDRAM operation parameters. It is a part of the SDRAM power-up sequence. Load mode register uses the address bus of the SDRAM as data input. The power-up sequence is enabled by writ-

ing 1 to the SDPSS bit in the SDCTL register, subsequent SDRAM accesses initiate the power-up sequence. The exact order of the power-up sequence is determined by the SDPM bit of the SDCTL register.

The load mode register command initializes the following parameters.

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

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

Bank Activation

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

Single Precharge

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

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

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 3-5](#) and [Figure 3-6](#) show the SDRAM write and read timing of the processors.

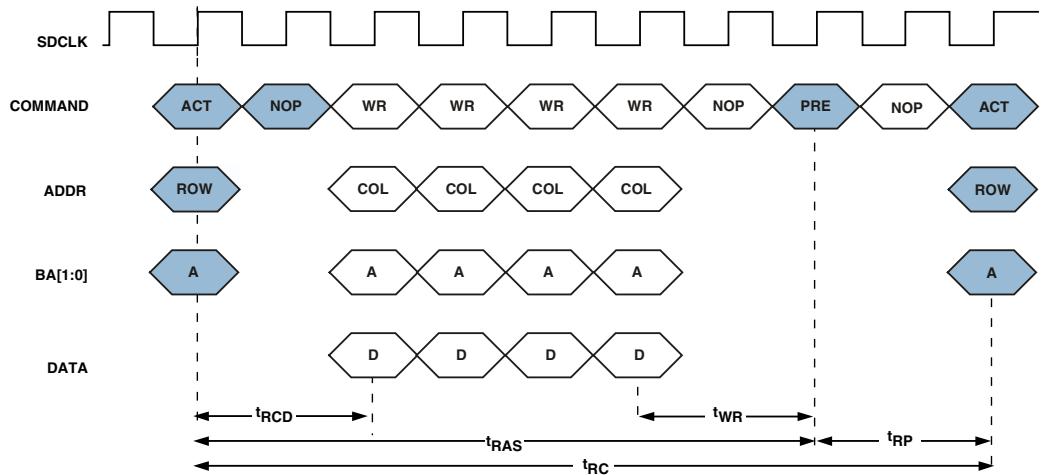


Figure 3-5. Write Timing Diagram

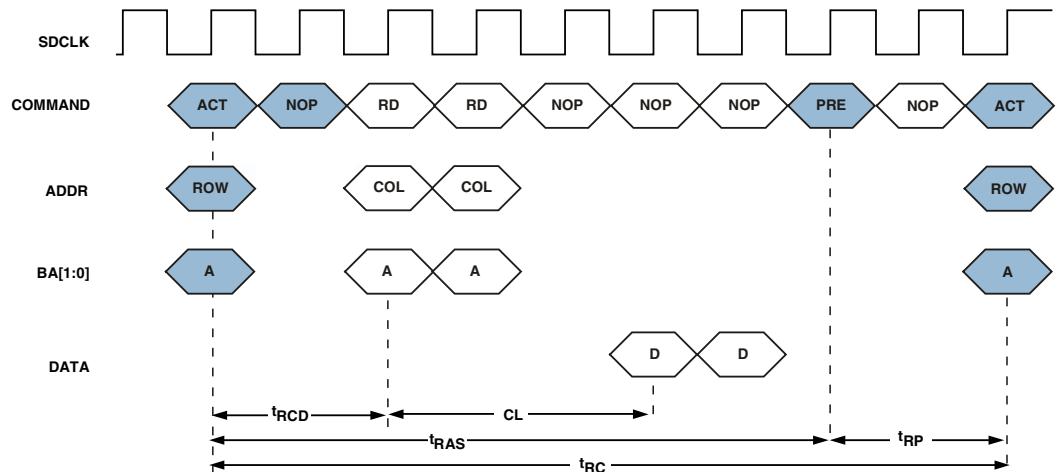


Figure 3-6. Read Timing Diagram

Auto-Refresh

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

Before executing the auto-refresh command, the 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.

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 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.

Command Truth Table

[Table 3-5](#) provides the bit states of the SDRAM for specific SDRAM commands. Note that an X means do not care.

Table 3-5. SDRAM Pin States During SDC Commands

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

Address Mapping

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

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

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

Address Translation Options

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

Page Interleaving Map

Programming the SDADDRMODE bit to 1 selects the page interleaving scheme. In this scheme consecutive pages fall in consecutive banks. The bank address bits follow the most significant column address bits. This is shown in [Figure 3-7](#).

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



Page interleaving is not supported with 2 bank devices.

Bank Interleaving Map

Programming the SDADDRMODE bit to 0 selects the bank interleaving scheme. In this scheme consecutive pages sit in the same bank. The bank address bits follow the most significant row address bits. This is shown in [Figure 3-7](#).

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



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

CORE ADDRESS MAPPING, TO ROW, COLUMN ADDRESSES (Page Interleaving, SDADDRMODE=1)

31

0

Unused	Row Address	Bank Address	Column Address
--------	-------------	--------------	----------------

CORE ADDRESS MAPPING, TO ROW, COLUMN ADDRESSES (Bank Interleaving, SDADDRMODE=0)

31

0

Unused	Bank Address	Row Address	Column Address
--------	--------------	-------------	----------------

Figure 3-7. Core Address Mapping—Page and Bank Interleaving



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

Address Width Settings

Address width settings can be configured as shown in [Table 3-11](#).

Table 3-6. External Memory Address Bank Decoding

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

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

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

Column Address Width (SDCAW). The SDRAM memory control register also includes external bank specific programmable parameters. The external bank can be configured for a different SDRAM size. The SDRAM controller determines the internal SDRAM page size from the X16DE and SDCAW parameters. Page sizes of 128, 256, 512, 1K, 2K words are supported.

16-Bit Address Mapping

Even if the external data width is 16 bits, the processor supports only 32-bit data accesses. If X16DE is enabled (=1) the 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.

In the following sections and in [Table 3-7](#) through [Table 3-10](#), the mapping of internal addresses to the external addresses is discussed. The mapping of the addresses depends on the address mode (SDAD-DRMODE) on row address width (SDRAW), and on column address width (SDCAW).



The X16DE bit must always be set.

For example, if the processor core requests address 0x200–0000 for a 32-bit access, the SDC performs two 16-bit accesses at 0x400–0000 and 0x400–0001, using $\overline{MS0}$ to get one 32-bit data word. The column and row addresses seen by 16-bit SDRAMs is shown in [Table 3-7](#) where

SDADDRMODE = 1, X16DE = 1, SDRAW2–0 = 101 (13 bits), and SDCAW1–0 = 10 (10 bits).

Table 3-7. Page Interleaving Map (1K Page Size)

Pin	Column Address	Row Address	Bank Address	Pins of SDRAM
A[18]			IA[10]	BA[1]
A[17]			IA[9]	BA[0]
A[13]				
A[12]		IA[23]		A[12]
A[11]		IA[22]		A[11]
SDA10	1'b0	IA[21]		A[10]
A[9]	IA[8]	IA[20]		A[9]
A[8]	IA[7]	IA[19]		A[8]
A[7]	IA[6]	IA[18]		A[7]
A[6]	IA[5]	IA[17]		A[6]
A[5]	IA[4]	IA[16]		A[5]
A[4]	IA[3]	IA[15]		A[4]
A[3]	IA[2]	IA[14]		A[3]
A[2]	IA[1]	IA[13]		A[2]
A[1]	IA[0]	IA[12]		A[1]
A[0]	1/0	IA[11]		A[0]

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

Table 3-8 where SDADDRMODE = 0, X16DE = 1, SDRAW2-0 = 100 (12 bits), and SDCAW1-0 = 11 (11 bits).

Table 3-8. Page Interleaving Map (2K Page Size)

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

In [Table 3-9](#), SDADDRMODE = 1, X16DE = 1, SDRAW2-0 = 101 (13 bits), and SDCAW = 10 (10 bits).

Table 3-9. Bank Interleaving Map (1K Page Size)

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

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

Table 3-10 where SDADDRMODE = 0, X16DE = 1, SDRAW2-0 = 100 (12 bits), and SDCAW1-0 = 11 (11 bits).

Table 3-10. Page Interleaving Map (2K Page Size)

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]

Refresh Rate Control

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.

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.

- $f_{SDCLK} = 133$ MHz

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

- $t_{REF} = 64$ ms
- NRA = 8192 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.

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.



Notice that some SDRAM vendors use separate timing specifications for the row active time (t_{RC}) and row refresh time (t_{RFC}).

The controller does ignore the t_{RFC} spec. For auto-refresh, it uses the equation $t_{RC} = t_{RAS} + t_{RP}$. However since both timing specifications must meet (especially for extended temperature range) the modification of t_{RAS} specification resolves the timing equation without performance degradation ($t_{RFC} = t_{RAS} + t_{RP}$).

Internal SDRAM Bank Access

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

Single Bank Access

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

Multibank Access

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



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

Any first access to 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 3-8](#).

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



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

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

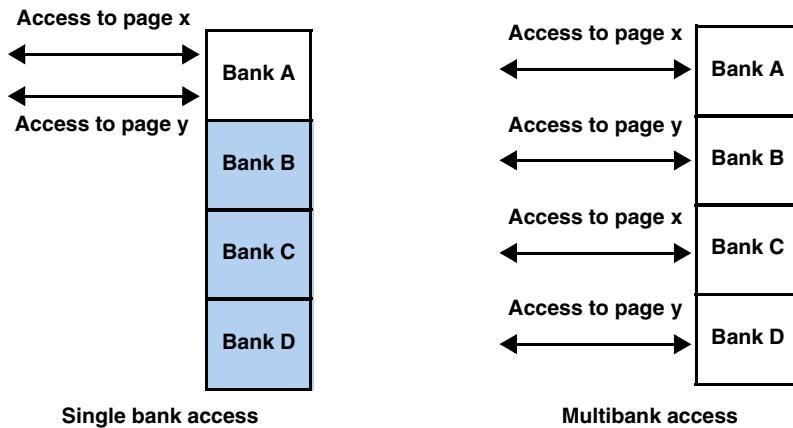


Figure 3-8. Single Versus Multibank Access

Multi Bank Operation with Data Packing

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

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

```
0x2000000    logical start address int bankA  
0x2000FF     logical end address int bankA  
  
0x3000000    logical start address int bankB  
0x3000FF     logical end address int bankB  
  
0x4000000    logical start address int bankC  
0x4000FF     logical end address int bankC  
  
0x5000000    logical start address int bankD  
0x5000FF     logical end address int bankD
```

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

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

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

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

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

Timing Parameters

The controller requires many timing settings in order to correctly access the SDRAM devices. Those that are user configurable can be found in “[SDRAM Registers](#)” on page A-51.

Fixed Timing Parameters

The timing specifications below are fixed by the controller.

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

Data Mask

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

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

Resetting the Controller

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

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



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

Operating Modes

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

Parallel Connection of SDRAMs

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

- 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 as shown in [Figure 3-10](#).

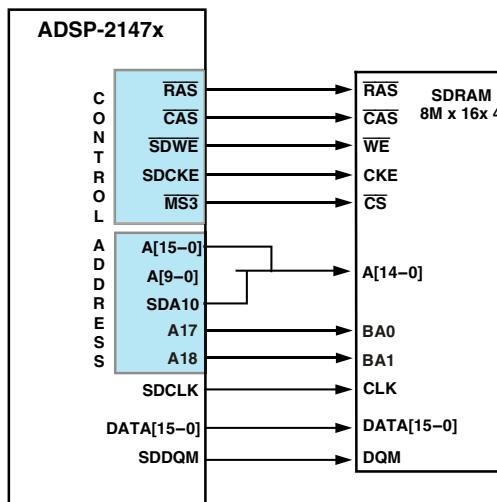


Figure 3-9. Single Processor System With SDRAM

Buffering Controller for Multiple SDRAMs

If using multiples SDRAMs or modules, the capacitive load will exceed the controller's output drive strength. In order to bypass this problem an external latch can be used for decoupling by setting the `SDBUF` (bit 23). This adds a cycle of data buffering to read and write accesses. An example single processor system is shown in [Figure 3-10](#).

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

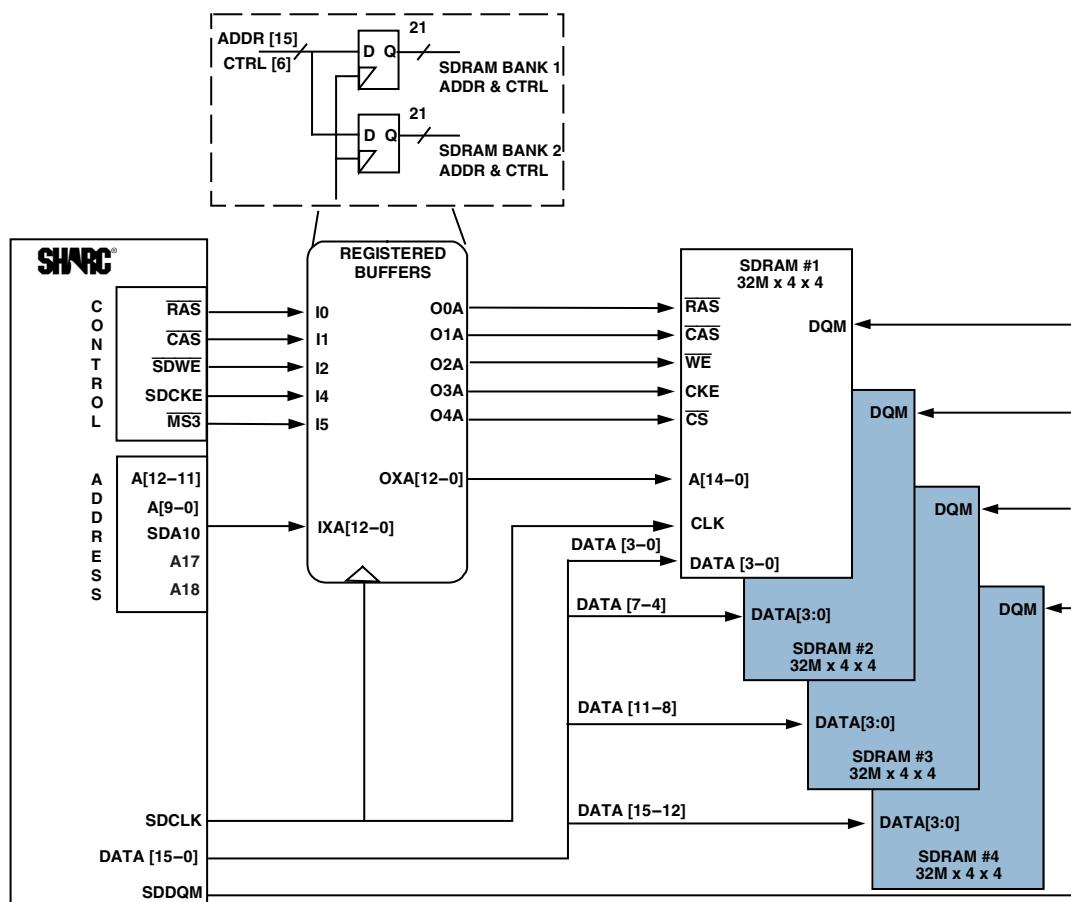


Figure 3-10. Uniprocessor System With Multiple Buffered SDRAM Devices

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 according to the core’s DAG modifier or the DMA’s modify parameter register.

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

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

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



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

Core Accesses

Any break of sequential reads of full page accesses can cause a throughput loss due to a maximum of four extra reads (eight 16-bit reads). Listing 3-1 shows how to achieve maximum throughput using core accesses. Any cycle between consecutive reads to an SDRAM address results in non-sequential reads.

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

Listing 3-1. Maximum Throughput Using Sequential Reads

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

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

Without read optimization, 512 reads use 6144 processor cycles if all of the reads are on the same page. With read optimization ([Listing 3-2](#)), 512 reads take 7168 cycles, due to the breaking of sequential reads.

Listing 3-2. Interrupted Reads With Read Optimization

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

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

DMA Access

[Listing 3-3](#) shows an example of external port DMA using read optimization.

Listing 3-3. EPDMA With Read Optimization

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

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

Notes on Read Optimization

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

Self-Refresh Mode

This mode causes refresh operations to be performed internally by the SDRAM, without any external control. This means that the 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 deasserts the `SDCKE` pin and puts the SDRAM in self-refresh mode if no access is currently underway. The SDRAM remains in self-refresh mode for at least t_{RAS} and until an internal access (read/write) to SDRAM space occurs.

Self-refresh exit—When any SDRAM access occurs, the 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.

Forcing SDRAM Commands

The SDC does have some specific bits which can be used to aid in debug and in specific system solutions.

Force Precharge All

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

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

DDR2 DRAM Controller (ADSP-2146x)

Force Load Mode Register

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

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

Force Auto-Refresh

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

DDR2 DRAM Controller (ADSP-2146x)

The DDR2 DRAM controller on ADSP-2146x processors enable a transfer of data to and from synchronous DDR2 DRAM. It supports a glueless interface with four external banks, controlled by the memory chip select pins (DDR2_CS3-0), of standard DDR2 DRAMs of 256 Mbit to 2 Gbit with configurations x8 and x16.

Features

The features of the DDR2 DRAM controller are listed below.

- I/O width 16-bits, I/O supply 1.8 V
- Supports DDR2-400 of 256M bit, 512M bit, 1G bit and 2G bit with configurations of x8 and x16
- Supports page sizes of 512, 1K, 2K, and 4K words

- Supports 4 and 8 bank DDR2 devices
- Variable memory address map (bank or page interleaving)
- Supports a maximum of 2G bit through the external DDR2 bank (64M words x 32)
- Supports up to 254M words of DDR2 memory
- Burst mode of 4 (BL = 4) with sequential burst type
- Open page policy—any open page is closed only if a new access in another page of the same bank occurs
- Supports multibank operation within the DDR2
- Uses a programmable refresh counter to coordinate between varying clock frequencies and the DDR2's required refresh rate
- Provides multiple timing options to support additional buffers between the SHARC-2146x and DDR2
- Supports self-refresh mode and precharge power-down to reduce power consumption
- Predictive data accesses for higher read data throughput (read optimization)
- Supports posted CAS additive latency (AL)
- Built in DLL to align DDR2_DATA and DDR2_DQS
- Supports programmable on-die termination (ODT) with the ODT pin
- Supports external instruction fetch in bank 0 for ISA and VISA operation
- Supports 64-bit SIMD mode by the core

DDR2 DRAM Controller (ADSP-2146x)

- Supports dual data instruction type 1
- Parallel access between DDR2 and AMI possible (no multiplexed pins)

Pin Descriptions

The pins used by the external memory interface are described in the *ADSP-2146x SHARC Processor Data Sheet*. Additional information on pin multiplexing can be found in “[Pin Descriptions](#)” on page 23-2.

Functional Description

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

Therefore, the architecture is enhanced into three blocks in order to fulfill the high speed constraints. One block is the DDR2 controller which interfaces to the core or DMA containing the state machine to provide the various commands to the DDR2 memory. Another block is the important DDR2 DLL circuit connected to the DDR2 controller and the final block contains the data capture (I/O pads).

The DDR2 DLL acts as an on-chip interface between the on-chip DDR2 controller and the off-chip DDR2 DRAM to meet the timing requirements of either block. [Figure 3-11](#) is a representation of part a system and shows the interaction of the DDR2 DLL with the controller and the external memory. There is one DDR2 DLL1–0 block for every set of 8 bits of data (`DDR2_DATA`), data strobe (`DDR2_DQS`), and data mask (`DDR2_DM`).

It should be noted that the DDR2 DLL acts as an interface for only the signals mentioned above and the clock output (`DDR2_CLK`) to the DDR2, it does not operate on the `DDR2_ADDR`, `DDR_CS` and command lines (`DDR2_RAS`, `DDR2_CAS`, `DDR2_WE`).

During a memory read, the data from the DDR2 memory (SSTL-18 level) is converted to the core voltage logic level inside the DDR2 memory I/O pads. This is captured by the DDR2 DLL using precise delays on the data strobe (`DDR2_DQS`) line in accordance with the JEDEC specification and provided to the controller. During a memory write, the data strobe (`DDR2_DQS1-0`) signal to the DDR2 is delayed by a fixed amount. Note there are stringent timing requirements between the DDR2 DLL and the controller.

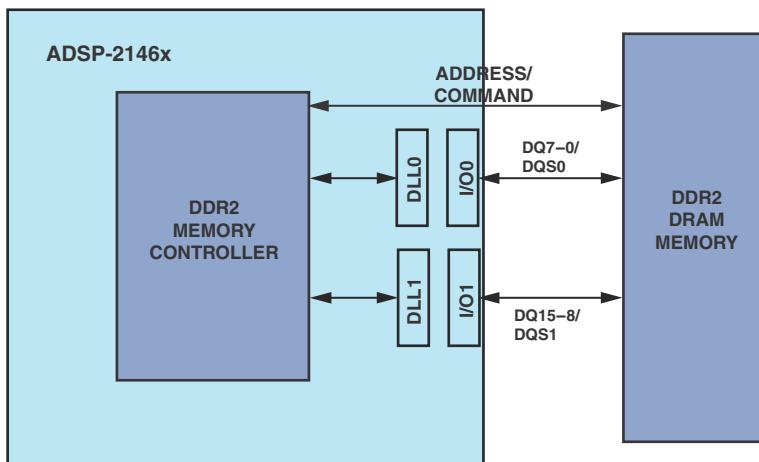


Figure 3-11. DDR2 Controller

The DDR2 DLL controls the setting of the delay for the data capture. The delay elements and DLL in the block are sensitive to process, voltage and temperature variations as well as to the operating frequency.

The DDR2 controller enables either the read or the write path in the memory I/O. Read data is sent by the DDR2 DRAM on both the rising

DDR2 DRAM Controller (ADSP-2146x)

and falling edges of the `DDR2_DQS` signal. The read data is captured by the DDR2 DLL using a delayed `DQS` that is phase shifted by approximately 90 degrees for the positive edge data and by approximately 90 degrees for the negative edge data. These delays are precisely generated using the internal DDR2 DLL circuit.

The captured data is sent out, corresponding to the data launched by the DRAM with the positive edge and negative edge of `DDR2_DQS` respectively. Both data buses are internally retimed such that they can be captured directly by the controller on the positive edge of `DDR2_CLK`, irrespective of the arbitrary phase relation that may exist between `DDR2_CLK` and the `DDR2_DQS`. During initial operation (external bank calibration), the DLL determines the phase difference between the `DDR2_CLK` and `DDR2_DQS` and retimes the data captured accordingly.

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

The configuration is programmed in the `DDR2CTL5-0` registers. The DDR2 controller can hold off the processor core or DMA controller with an internally connected acknowledge signal, as controlled by refresh, or page miss latency overhead. A programmable refresh counter is provided which generates background auto-refresh cycles at the required refresh rate based on the clock frequency used. The refresh counter period is specified using the `RDIV` field in the DDR2 refresh rate control register ([“Refresh Rate Control Register \(DDR2RRC\)” on page A-36](#)).

The DDR2C uses burst length 4 ($BL = 4$) for read and write operations. This requires the DDR2C to post only the first read or write address on the bus, all subsequent sequential address are posted by the DDR2 internal burst counter.

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



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

DDR2 Commands

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

Load Mode Register

This command initializes DDR2 operation parameters and controls part of the power-up sequence, initiated by writing 1 to the `DDR2PSS` bit in the DDR2 memory control register (`DDR2CTL2`). Values written into the `DDR2CTL0` register are loaded into the `MR` register during power up. The command uses the address bus of the DDR2 for data input.

The load mode register command initializes the following parameters.

- Bits 2–0 – Burst length = 4
- Bit 3 – Wrap type (BT) = sequential (always zero)

DDR2 DRAM Controller (ADSP-2146x)

- Bits 6–4 – CAS latency, programmable in the `DDR2CTL0` register
- Bit 7 – Always 0
- Bit 8 – reset DLL (DDR2 device)
- Bits 11–9 = Reserved
- Bits 13–12 = Always zero
- Bits 15–14 (00 – selects mode register)

While executing this command, the unused address pins are set to zero. During the first `DDR2_CLK` cycle following the command, the controller issues a `NOP` command. This command can also be triggered by setting the `FLMR` bit in the `DDR2CTL0` register.



To automatically start the power-up sequence, (no dummy access are required) set the `DDR2PSS` bit (=1).

Load Extended Mode Register

This command initializes DDR2 operation parameters (other than those controlled by mode register). This command is a part of the power-up sequence, initiated by writing 1 to the `DDR2PSS` bit in the DDR2 memory control register (`DDR2CTL0`). This command uses the address bus of the controller for data input.

Values written into `DDR2CTL3` register are loaded into the `EMR` register during power up. The command initializes the following parameters:

1. Bit 0 – DLL enable/disable
2. Bit 1 – ODS (output drive strength—reduced/full)
3. Bits 2, 6 = Rtt value (ODT feature)
4. Bits 5–3 = Additive latency from 0 to 5
5. Bits 9–7 = Always zero

6. Bit 10 – Differential $\overline{\text{DDR2_DQS}}$ enable/disable
7. Bit 12 – Output buffer enable/disable
8. Bits 15–14 = 01 for EMR1

The command can also be triggered by setting the FEMR bit in the DDR2CTL0 register.

Load Extended Mode Register 2

Values written into the DDR2CTL4 register are loaded into the EMR2 register as is during power up.

1. Bits 13–0 = always zero
2. Bits 15–14 = 10 for EMR2

Load Extended Mode Register 3

Values written into the DDR2CTL5 register are loaded into the EMR3 register during power up.

1. Bits 13–0 = always zero (OCD exit)
2. Bits 15–14 = 10 for EMR3



The DDR2 controller does not support off-chip driver (OCD) calibration. Also note that all mode registers must be programmed since the default settings in the DDR2 device are not defined.

Bank Activation

This command is required if the next data access is on a different page in the same internal bank or in a different internal bank that is in an idle state. The controller executes the pre-charge command, followed by a bank activate command, to activate the page in the desired DDR2 internal bank. The controller is able to open up to eight pages at the same time in

DDR2 DRAM Controller (ADSP-2146x)

different internal banks. For 8 banked devices, the controller does follow the t_{FAW} specification.

Precharge

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

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

Precharge All

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

Burst Read

The burst read command is initiated by having $\overline{DDR2_CS}$ and $\overline{DDR2_CAS}$ low while holding $DDR2_RAS$ and $DDR2_WE$ high at the rising edge of the clock. The address inputs determine the starting column address for the burst. The delay from the start of the command to when the data from the first cell appears on the outputs is equal to the value of the read latency (RL).

The data strobe output ($DDR2_DQS$) is driven low one clock cycle before valid data ($DDR2_DATA$) is driven onto the data bus ([Figure 3-12](#)). The first bit of the burst is synchronized with the rising edge of the data strobe ($DDR2_DQS$).

Each subsequent data-out appears on the $DDR2_DATA$ The first bit of the burst is synchronized with the rising edge of the data strobe ($DDR2_DQS$).

Each subsequent data-out appears on the `DDR2_DATA` pin in phase with the `DDR2_DQS` signal in a source synchronous manner.

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

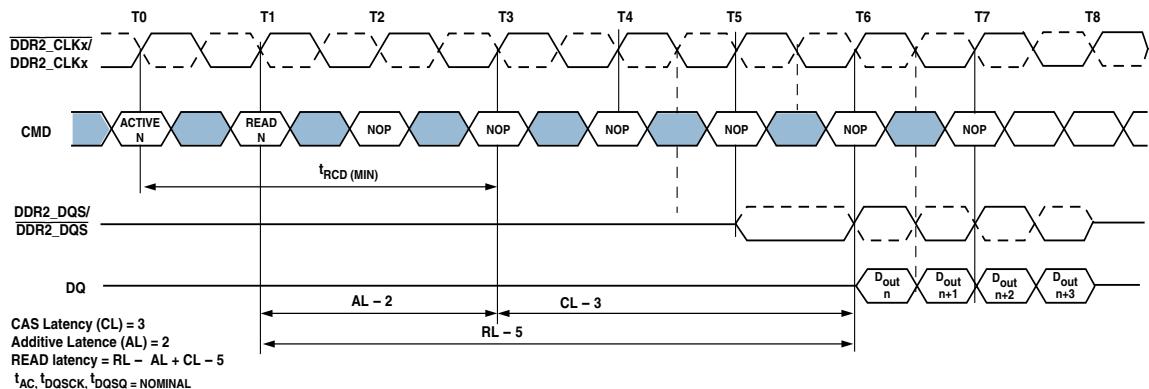


Figure 3-12. Burst Read

Burst Write

The burst write command, shown in [Figure 3-13](#), is initiated by having `DDR2_CS`, `DDR2_CAS` and `DDR2_WE` pins low while holding `DDR2_RAS` high at the rising edge of the clock. The address inputs determine the starting column address. Write latency (WL) is defined by a read latency (RL) minus one and is equal to $(AL + CL - 1)$ and is the number of clocks of delay that are required from the time the write command is registered to the clock edge associated to the first `DDR2_DQS` strobe.

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

DDR2 DRAM Controller (ADSP-2146x)

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

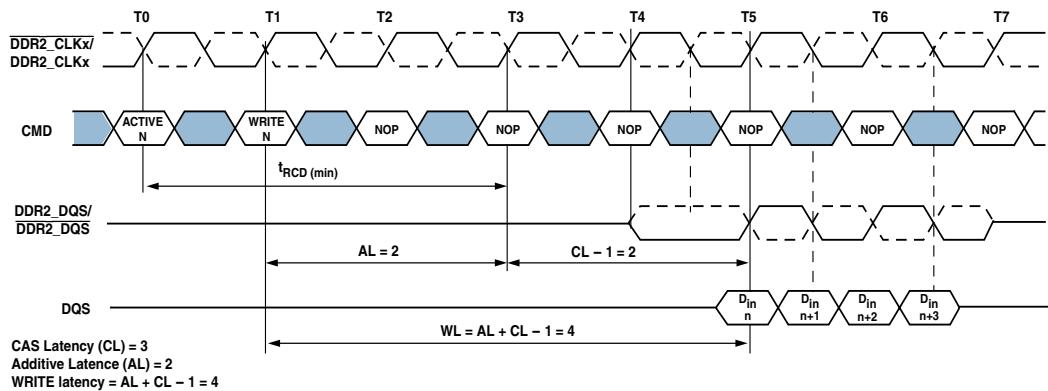


Figure 3-13. Burst Write

Auto-Refresh

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

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

Self-Refresh Entry

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

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

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

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

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



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

Self-Refresh Exit

The DDR2 remains in self-refresh mode for at least t_{RAS} and until an internal access to DDR2 space occurs. When exiting from self-refresh

DDR2 DRAM Controller (ADSP-2146x)

mode programs need to consider if this occurs during a read or write. If exiting during a read, additional latency occurs because the on-chip DLL needs to be locked again.

When an internal access occurs (or with careful software control) the SREF_EXIT bit is set in DDR2CTL0 register and the controller:

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

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

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

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

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

Precharge Power-Down Entry

The DDR2 controller supports DDR2 precharge power down mode. In this mode, the DDR2 device's DLL is frozen to maximize power consumption.

When the DIS_DDR2CKE bit is set to 1 and the DDR2 controller enters an idle state, it issues a pre-charge command (if necessary) and then, after

meeting the required timing specifications, pulls down the `DDR2CKE` signal. If an internal access is pending, the controller delays entering the power-down mode until it completes the pending DDR2 access and any subsequent pending access requests.

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

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



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

This mode is useful if the DDR2 operation is idling only for a *short* period of time. This time is limited by the JEDEC spec and is typically $9 \times t_{REFI}$. If for example $t_{REFI} = 7.8\mu s$ the maximum power-down time is $9 \times 7.8\mu s = 70\mu s$.

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

Precharge Power-down Exit

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

No Operation/Command Inhibit

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

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

Address Mapping

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

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

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

Address Translation Options

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

Page Interleaving Map

Programming the `DDR2ADDRMODE` bit to 0 selects the page interleaving scheme. In this scheme consecutive pages fall in consecutive banks. The bank address bits follow the most significant column address bits. This is shown in [Figure 3-14](#).

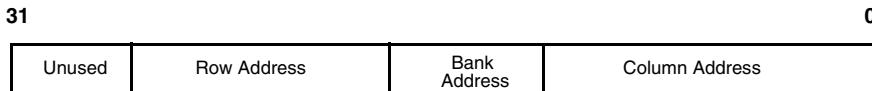


Figure 3-14. Core Address Mapping to Row, Column Addresses (Page)

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

Bank Interleaving Map

Programming the `DDR2ADDRMODE` bit to 1 selects the bank interleaving scheme. In this scheme consecutive pages sit in the same bank. The bank address bits follow most significant row address bits. This is shown in [Figure 3-15](#).

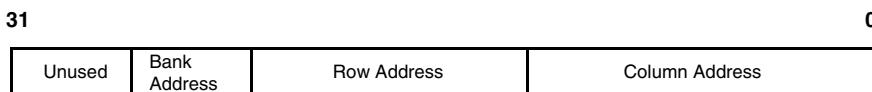


Figure 3-15. Core Address Mapping to Row, Column Addresses (Bank)

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

Address Width Settings

Number Internal Banks (DDR2BC). The controller assumes the DDR2 is comprised of eight bank devices. However, DDR2 can use four bank devices by not connecting the `ADDR18` pin and programming the `DDR2BC`

DDR2 DRAM Controller (ADSP-2146x)

bits in the `DDR2CTL0` register. The bank address width is three bits as shown in [Table 3-11](#).

Table 3-11. External Memory Address Bank Decoding

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

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

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



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

16-Bit Address Mapping

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

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



The X16DE bit must always be set.

Address Map Tables

The row address and column address mappings for 16-bit addresses are shown in [Table 3-12](#) through [Table 3-15](#). The row, bank and column addresses are multiplexed to the A18–A0 pins of the processor.

[Table 3-12](#) through [Table 3-15](#) also show the mapping of the internal address [IA] to the external address. The mapping of the address depends on row address width, column address width, the number of internal banks, and the external I/O width.

[Table 3-11](#) shows DDR2ADDRMODE = 0, DDR2RAW = 100 (12), DDR2CAW = 10 (10), DDR2BC = 10.

Table 3-12. 16-bit Address Mapping (8 Banks, Page Interleaving)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_BA2			IA[11]	BA[2]
DDR2_BA1			IA[10]	BA[1]
DDR2_BA0			IA[9]	BA[0]
DDR2_ADDR[12]				A[12]
DDR2_ADDR[11]		IA[23]		A[11]
DDR2_ADDR[10]		IA[22]		A[10]
DDR2_ADDR[9]	IA[8]	IA[21]		A[9]
DDR2_ADDR[8]	IA[7]	IA[20]		A[8]
DDR2_ADDR[7]	IA[6]	IA[19]		A[7]
DDR2_ADDR[6]	IA[5]	IA[18]		A[6]
DDR2_ADDR[5]	IA[4]	IA[17]		A[5]

DDR2 DRAM Controller (ADSP-2146x)

Table 3-12. 16-bit Address Mapping (8 Banks, Page Interleaving) (Cont'd)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_ADDR[4]	IA[3]	IA[16]		A[4]
DDR2_ADDR[3]	IA[2]	IA[15]		A[3]
DDR2_ADDR[2]	IA[1]	IA[14]		A[2]
DDR2_ADDR[1]	IA[0]	IA[13]		A[1]
DDR2_ADDR[0]	1/0	IA[12]		A[0]

Table 3-12 shows DDR2ADDRMODE = 0, DDR2RAW = 100 (12), DDR2CAW = 11 (11), DDR2BC = 01(four banks).

Table 3-13. 16-bit Address Mapping (4 Banks, Page Interleaving)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_BA1			IA[11]	BA[1]
DDR2_BA0			IA[10]	BA[0]
DDR2_ADDR[13]				
DDR2_ADDR[12]				A[12]
DDR2_ADDR[11]	IA[9]	IA[23]		A[11]
DDR2_ADDR[10]		IA[22]		A[10]
DDR2_ADDR[9]	IA[8]	IA[21]		A[9]
DDR2_ADDR[8]	IA[7]	IA[20]		A[8]
DDR2_ADDR[7]	IA[6]	IA[19]		A[7]
DDR2_ADDR[6]	IA[5]	IA[18]		A[6]
DDR2_ADDR[5]	IA[4]	IA[17]		A[5]
DDR2_ADDR[4]	IA[3]	IA[16]		A[4]
DDR2_ADDR[3]	IA[2]	IA[15]		A[3]
DDR2_ADDR[2]	IA[1]	IA[14]		A[2]

Table 3-13. 16-bit Address Mapping (4 Banks, Page Interleaving) (Cont'd)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_ADDR[1]	IA[0]	IA[13]		A[1]
DDR2_ADDR[0]	1/0	IA[12]		A[0]

[Table 3-14](#) shows DDR2ADDRMODE = 1, DDR2RAW = 100 (12), DDR2CAW = 10 (10), DDR2BC = 10 (eight banks).

Table 3-14. 16-bit Address Mapping (8 Banks, Bank Interleaving)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_BA2			IA[23]	BA[2]
DDR2_BA1			IA[22]	BA[1]
DDR2_BA0			IA[21]	BA[0]
DDR2_ADDR[12]				A[12]
DDR2_ADDR[11]		IA[20]		A[11]
DDR2_ADDR[10]		IA[19]		A[10]
DDR2_ADDR[9]	IA[8]	IA[18]		A[9]
DDR2_ADDR[8]	IA[7]	IA[17]		A[8]
DDR2_ADDR[7]	IA[6]	IA[16]		A[7]
DDR2_ADDR[6]	IA[5]	IA[15]		A[6]
DDR2_ADDR[5]	IA[4]	IA[14]		A[5]
DDR2_ADDR[4]	IA[3]	IA[13]		A[4]
DDR2_ADDR[3]	IA[2]	IA[12]		A[3]
DDR2_ADDR[2]	IA[1]	IA[11]		A[2]
DDR2_ADDR[1]	IA[0]	IA[10]		A[1]
DDR2_ADDR[0]	1/0	IA[9]		A[0]

DDR2 DRAM Controller (ADSP-2146x)

Table 3-15 shows DDR2ADDRMODE = 1, DDR2RAW = 100 (12), DDR2CAW = 11 (11), DDR2BC = 01(four banks).

Table 3-15. 16-bit Address Mapping (4 Banks, Bank Interleaving)

SHARC Pin	Column Address	Row Address	Bank Address	DDR2 Pin
DDR2_BA1			IA[23]	BA[1]
DDR2_BA0			IA[22]	BA[0]
DDR2_ADDR[13]				
DDR2_ADDR[12]				A[12]
DDR2_ADDR[11]	IA[9]	IA[21]		A[11]
DDR2_ADDR[10]		IA[20]		A[10]
DDR2_ADDR[9]	IA[8]	IA[19]		A[9]
DDR2_ADDR[8]	IA[7]	IA[18]		A[8]
DDR2_ADDR[7]	IA[6]	IA[17]		A[7]
DDR2_ADDR[6]	IA[5]	IA[16]		A[6]
DDR2_ADDR[5]	IA[4]	IA[15]		A[5]
DDR2_ADDR[4]	IA[3]	IA[14]		A[4]
DDR2_ADDR[3]	IA[2]	IA[13]		A[3]
DDR2_ADDR[2]	IA[1]	IA[12]		A[2]
DDR2_ADDR[1]	IA[0]	IA[11]		A[1]
DDR2_ADDR[0]	1/0	IA[10]		A[0]

Refresh Rate

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

The delay (in number of `DDR2_CLKx` cycles) desired between consecutive refresh counter time-outs must be written to the `RDIV` field. A refresh counter time-out triggers an auto-refresh command to the external DDR2 bank. Write the `RDIV` value to the `DDR2RRC` register before the DDR2 power-up sequence is triggered. Change this value only when the DDR2 controller is idle.

To calculate the value that should be written to the `DDR2RRC` register, use the following equation:

$$\text{RDIV} = (\text{DDR2_CLKx} \times t_{\text{REFI}}) - (t_{\text{RAS}} + t_{\text{RP}}) \text{ where:}$$

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

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

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

$$\text{DDR2_CLKx} = 200 \text{ MHz}$$

$$t_{\text{REFI}} = 7.8 \mu\text{s}$$

DDR2 DRAM Controller (ADSP-2146x)

$t_{RAS} = 9$ cycles

$t_{RP} = 3$ cycles

The equation for RDIV yields:

$$RDIV = (200 \times 10^6 \times 7.8 \times 10^{-6}) - (9 + 3) = 1548 \text{ clock cycles.}$$

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

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

Data Mask

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

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



The DDR2_DM1-0 pins are useful for performance monitoring during write commands. Every time these signals are asserted indicates the controller masks unwanted data writes causing performance penalties. For reads, the controller simply does not latch the data from the burst.

Resetting the Controller

Like any other peripheral, the DDR2 controller can be reset by hard- or a soft reset. Both reset modes pull the DDR2_CKE pin asynchronously low. Since DDR2_CKE drops asynchronously and the PLL goes into bypass mode

(hardware reset) immediately after reset, timing parameter cannot be met, causing data loss. The DDR2 device must be re-initialized and the DDR2 DLL must be re locked to use the DDR2 again.



Running reset (`RESETOUT` pin as an input) does not reset the DDR2 controller.

Disabling the Controller

If the DDR2 interface is not used, the following bits should be configured. This is required get maximum power reduction.

- In the `DDR2CTL0` register, set (=1) the following bits: `DIS_DDR2CTL`, `DIS_DDR2CLK1` and `DIS_DDR2CKE` to disable the controller and its I/O pads.
- In the `DDR2PADCTL0` register (bits 9, 19 and 29) and `DDR2PADCTL1` register (bits 9 and 19), set (=1) all the `PWD` bits to power-down the pad receivers.

Initialization Sequence

After the `DDR2PSS` bit is set in the `DDR2CTL0` register, the DDR2 controller starts the power-up initialization sequence which occurs in the following order. Note that this procedure is performed by the DDR2 controller and user intervention is not required.

1. Brings `DDR2CKE` high, drive a NOP command.
2. Wait a minimum of 400 ns (with `NOP` or `DESELECT` commands).
3. Issue a precharge all command. Wait t_{RPA} period.
4. Issue a load EMR(2) command. Wait t_{MRD} period.
5. Issue a load EMR(3) command. Wait t_{MRD} period.
6. Issue a load EMR command. Wait t_{MRD} period.

DDR2 DRAM Controller (ADSP-2146x)

7. Issue a load MR command. Wait t_{MRD} period. Also trigger a counter (200 cycle counter)—any read command is issued only after this counter expires.
8. Issue a precharge all command. Wait t_{RPA} period.
9. Issue two or more auto refresh commands, with a t_{RFC} period in between each command. Wait t_{RFC} period.
10. Issue a load MR command with low to A8 (bit 8), to initialize operating parameters without resetting the DDR2 DLL. Wait for t_{MRD} period.
11. Issue a load EMR command.
12. Wait for t_{MRD} period.
13. Wait for the 200 cycle counter to expire before performing any read operation.
14. Start the calibration of the DLL within the processor's DDR2 controller.

The DDR2 is now ready for normal operation.

Initialization Time

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

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

Internal DDR2 Bank Access

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

Single Bank Access

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

Multibank Access

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



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

Any first access to DDR2 bank (A) forces an activate command before a read or write command. However, if any new access falls into the address space of the other banks (B, C, D, E, F, or H) the controller leaves bank (A) open and activates any of the other banks (B, C, D, E, F, or H). Bank (A) to bank (B) active time is controlled by t_{RRD} . This scenario is repeated until all eight banks (A–H) are opened and results in an effective page size of up to eight pages. This is because the absence of latency allows switching between these open pages (as compared to one page in only one bank at a time). Any access to any closed page in any opened bank (A–H) forces a precharge command only to that bank. If, for example, two external port DMA channels are pointing to the same internal DDR2 bank, this always forces precharge and activation cycles to switch between the different pages. However, if the two external port DMA channels are pointing to different internal DDR2 banks, there is no additional overhead. See [Figure 3-16](#).

DDR2 DRAM Controller (ADSP-2146x)

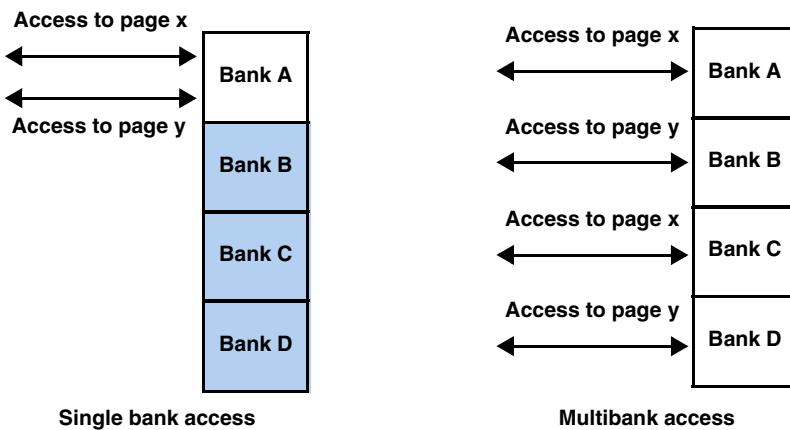


Figure 3-16. Single Versus Multibank Access

Force Activation Window

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

DDR2 devices support a new timing parameter called four active banks window (t_{FAW}). This is the minimum amount of time that must pass before more than four ACTIVE (ACT) commands may occur. It is acceptable to have more than 4 banks open simultaneously, but the additional ACT command(s) must be spaced out past the $t_{FAW}(\text{min})$ window. As shown in [Figure 3-17](#), t_{RCD} for the fourth opened bank is complete at T8. To satisfy $t_{FAW}(\text{min})$, the fifth ACT command cannot occur until T11.

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

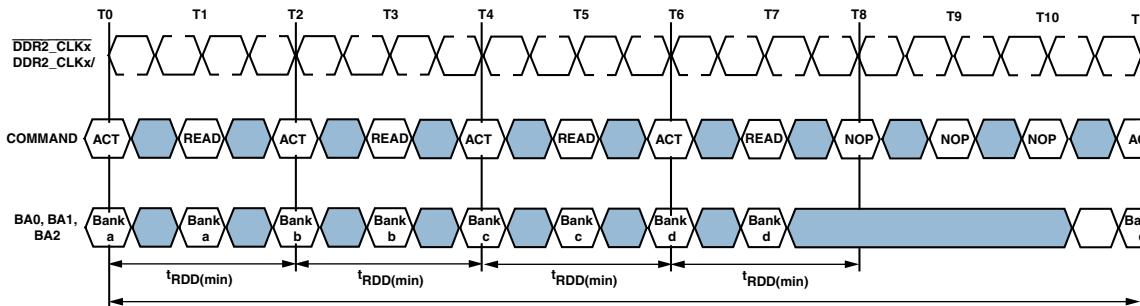


Figure 3-17. Bank Activation for a 8 Banked Device



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

Multi Bank Operation with Data Packing

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

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

Page Interleaving (DDR2ADDRMODE bit = 0):

```

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

0x200800 logical start address int bankE
0x2009FF logical end address int bankE

```

DDR2 DRAM Controller (ADSP-2146x)

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

Bank Interleaving (DDR2ADDRMODE bit = 1):

```
0x200000 logical start address int bankA  
0x2001FF logical end address int bankA  
0x600000 logical start address int bankB  
0x6001FF logical end address int bankB  
0xA00000 logical start address int bankC  
0XA001FF logical end address int bankC  
0xE00000 logical start address int bankD  
0XE001FF logical end address int bankD  
  
0x1200000 logical start address int bankE  
0x12001FF logical end address int bankE  
0x1600000 logical start address int bankF  
0x16001FF logical end address int bankF  
0x1A00000 logical start address int bankG  
0x1A001FF logical end address int bankG  
0x1E00000 logical start address int bankH  
0x1E001FF logical end address int bankH
```

Fixed Timing Parameters

The timing specifications below are fixed by the controller.

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

- t_{RFC} (row refresh cycle). Required delay time to refresh a single row. This parameter is fixed to $t_{RFC} = t_{RC}$ cycles.
- t_{XSNR} (exit self-refresh with non-read). Required delay to exit the self-refresh mode with a non read command. This parameter is fixed to $t_{XSNR} = t_{RFC} + 4$ cycles.
- t_{XSRD} (exit self-refresh with read). Required delay to exit the self-refresh mode with a read command. This parameter is fixed to $t_{XSRD} = 200$ cycles.

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

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

Operating Modes

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

Parallel Connection of DDR2s

To specify a DDR2 system, multiple possibilities are given based on the different memory sizes. For a 16-bit I/O capability, the following memory sizes can configured.

- 1 x 16-bit/page 512 words

DDR2 DRAM Controller (ADSP-2146x)

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

The DDR2's page size is used to determine the system you select. All three systems have the same external bank size, but different page sizes. Note that larger page sizes, allow higher performance but larger page sizes require more complex hardware layouts.



Even if connecting DDR2s in parallel, the DDR2C always considers the cluster as one external DDR2 bank because all address and control lines feed the parallel parts.

Buffering Controller for Multiple DDR2s

If using multiples DDR2s or modules, the capacitive load will exceed the controller's output drive strength. In order to bypass this problem an external register (SSTL18 class) can be used for decoupling by setting bit 24 in `DDR2CTL0` register. This adds a cycle of data buffering to read and write accesses.

Read Optimization

The best throughput numbers for reads are achievable only when the `DDR2OPT` bit in the `DDR2CTL0` register is set. To achieve better performance for reads, predictive addresses need to be given to the DDR memory. The predictive address given to the memory depends on the `DDR2MODIFY` bit setting. If the `DDR2MODIFY` value is 2 then the address + 2 is given predictively on the DDR address pins. Programs have the option whether to use read optimization or not.

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



For SIMD accesses, if optimization is enabled and the modifier is set to 2 (even if the modifier is changed, it remains at 2). The throughput is at maximum if optimization is enabled for sequential accesses. But in the case of non sequential accesses, throughput is affected by enabling optimization.

DDR2 Read Optimization

The predictive address given to the memory depends on the `DDR2MODIFY` bit values. For example, if the DAG modifier = 2, the `DDR2MODIFY` value should also be 2, in which case the address + 2 is the predictive value provided to the DDR2 address pins. Programs may choose to determine whether read optimization is used or not. If read optimization is disabled, then each read takes 7 cycles for a CAS latency of 3, even for sequential reads.

With read optimization enabled, 32 sequential reads, with offsets ranging from 0 to 15, take only 40 `DDR2CLK` cycles. Read optimization should not be enabled while reading at the external bank boundaries. For example, if `DDR2MODIFY` = 1, then 32 locations in the boundary of the external banks should not be used. These locations can be used without optimization enabled. If `DDR2MODIFY` = 2, then 64 locations cannot be used at the boundaries of the external bank (if it is fully populated).



By default, the read optimization is enabled (`SDROPT` = 1) with a modifier of 1 (`DDR2MODIFY` = 1). Read optimization assumes that the DDR2 pointer has a constant modifier. For non-sequential accesses, turning off optimization provides better results.

Core Accesses

Any break of sequential reads of full page accesses can cause a throughput loss due to a maximum of eight extra reads. Listing 3-4 shows how to achieve maximum throughput using core accesses. Any cycle between consecutive reads to an DDR2 address results in non-sequential reads.

DDR2 DRAM Controller (ADSP-2146x)

Listing 3-4. Maximum Throughput Using Sequential Reads

```
ustat1=dm(DDR2CTL0);
bit set ustat1 DDR2OPT|DDR2MODIFY1;
dm(DDR2CTL0)=ustat1;
nop;
I0 = DDR2_addr;
M0 = 1;
Lctr = 1024, do(PC,1) until lce;
R0 = R0 + R1, R0 = dm (I0, M0);
```

The example shows read optimization can be used efficiently using core accesses. All reads are on the same page and it takes 1044 cycles to perform 1024 reads.

Without read optimization, 1024 reads use 5125 processor cycles if all of the reads are on the same page, non-sequential reads takes 9220 cycles. With read optimization ([Listing 3-5](#)), 1024 reads take 10262 cycles, due to the breaking of sequential reads.

Listing 3-5. Interrupted Reads With Read Optimization

```
ustat1=dm(DDR2CTL0);
bit set ustat1 DDR2OPT|DDR2MODIFY2;
dm(DDR2CTL0)=ustat1;
nop;
I0 = DDR2_addr;
M0 = 2;
Lctr = 1024, do(PC,2) until lce;
R0 = R0 + R1, R0 = dm (I0, M0);
NOP;
```

Note the above mentioned cycles may vary based on different latency and timing parameters programmed.

DMA Access

[Listing 3-6](#) shows an example of external port DMA using read optimization.

Listing 3-6. EPDMA With Read Optimization

```

ustat1=dm(DDR2CTL0);
bit set ustat1 DDROPT|DDRMODIFY2;
dm(SDCTL)=ustat1;
nop;
r0=DFLSH;
dm(DMAC1)=r0;
r0=intmem;      dm(IIEP1)=r0;
r0=2;           dm(IMEP1)=r0;
r0=N;           dm(ICEP1)=r0;
r0=2;           dm(EMEP1)=r0;
r0=extmem;     dm(EIEP1)=r0;
r0=DEN;
dm(DMAC1)=r0;

```

Notes on Read Optimization

The core and the DMA engine take advantage of the major improvements during reads using read optimization. However, in situations where both the core and DMA need to read from different internal memory banks with different modifiers at the same time, programs need to choose whether or not to use optimization. Note that from a throughput prospective, external port arbitration also is a factor. A good rule is that the requester with the higher priority should have the same modifier as DDR2MODIFY. In other words, if DMA has a higher priority over the core, then the DMA modifier should match the DDR2MODIFY setting.

Self-Refresh Mode

This mode causes refresh operations to be performed internally by the DDR2, without any external control. This means that the SDC does not generate any auto-refresh cycles while the DDR2 is in self-refresh mode.

Self-refresh entry—Self-refresh mode is enabled by writing a 1 to the DDR2SRF bit of the DDR2 memory control register (DDR2CTL0). This deasserts the DDR2CKE pin and puts the DDR2 in self-refresh mode if no access

is currently underway. The DDR2 remains in self-refresh mode for at least t_{RAS} and until an internal access (read/write) to DDR2 space occurs.



The self-refresh entry command does automatically disable the DDR2 memory DLL. Therefore its release command (exit) requires additional stall cycles until the DLL has re-locked.

Self-refresh exit. When any DDR2 access occurs, the DDR2C asserts `DDR2CKE` high which causes the DDR2 to exit from self-refresh mode. The controller waits to meet the t_{XSNR} specification (exit with no read command) or the t_{XSRD} specification (exit with read command). Here is a significant difference; releasing with a read command requires 200 DDR2 cycles (since the memory DLL needs to re-read memory).

System clock during self-refresh mode. Note that the `DDR2CLK` is not disabled by the controller during self-refresh mode. However, software may disable the clocks by setting the `DIS_DDR2CTL` bit in the `DDR2CTL0` register. Programs should ensure that all applicable clock timing specifications are met before the transfer to DDR2 address space (which causes the controller to exit the self-refresh mode). If a transfer occurs to DDR2 address space when the `DIS_DDR2CTL` bit is cleared, an internal bus error is generated, and the access does not occur externally, leaving the DDR2 in self-refresh mode.

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

1. Set the `DDR2SRF` bit to enter self-refresh mode.
2. Poll the `DDR2SRA` bit in the DDR2 status register (`DDR2STAT`) to determine if the DDR2 has already entered self-refresh mode.
3. Optionally: set the `DIS_DDR2CTL` bit to freeze `DDR2_CLK`.
4. Optionally: clear the `DIS_DDR2CTL` bit to re-enable `DDR2_CLK`.

DDR2 access occurs and the DDR2 exits from self-refresh mode.



The minimum time between a subsequent self-refresh entry and exit command is the t_{RAS} cycle. If a self-refresh request is issued during any external port DMA, the DDR2 controller grants the request with the t_{RAS} cycle and continues DMA operation afterwards.

Single-Ended Data Strobe

DDR2 data strobe mode is specified for either single ended or differential mode, depending on the setting of the EMR register enable DDR2_DQS mode bit. The timing advantages of differential mode are realized in system design.

The method by which the DDR2 pin timing is measured is mode dependent. In single ended mode, timing relationships are measured relative to the rising or falling edges of `DDR2_DQS` crossing at VREF. In differential mode, these timing relationships are measured relative to the crosspoint of `DDR2_DQSS` and its complement, `DDR2_DQS`. This distinction in timing methods is guaranteed by design and characterization. When differential data strobe mode is disabled via the EMR register, the complementary pin, `DDR2_DQS`, must be tied externally to VSS through a $20\ \Omega$ to $10\ k\Omega$ resistor to insure proper operation.

On Die Termination (ODT)

The DDR2 controller contains a separate pin (`DDR2_ODT`) that controls on-die termination. By default this pin is deasserted. If during power-up, the `ODT` register field in the `DDR2CTL3` register is programmed with any Rtt value, the `ODT` pin is asserted after the power-up sequence has finished.

The level can be changed by forcing another power-up sequence which disables Rtt resistance in the `ODT` field. After completion, the `ODT` pin is deasserted. Note that the `ODT` pin control is independent on the DDR2 data access directions (read or write).

Additive Latency

Posted CAS operation is supported to make the command and data bus efficient for sustainable bandwidths in DDR2 SDRAM. In this operation, the DDR2 SDRAM allows a CAS read or write command to be issued immediately after the RAS bank activate command (or any time during the RAS-CAS-delay time, t_{RCD} , period). The command is held for the time of the additive latency (AL) before it is issued inside the device. The read latency (RL) is controlled by the sum of AL and the CAS latency (CL).

Therefore if a program wants to issue a read/write command before the $t_{RCD\min}$, then AL (greater than 0) must be written into EMR(1). The write latency (WL) is always defined as RL – 1 (read latency – 1) where read latency is defined as the sum of additive latency plus CAS latency (RL = AL + CL). Read or write operations using AL allow seamless bursts (refer to seamless operation timing diagram examples in read burst and write burst section). Note that the controller does support this feature, however the performance is regardless of the AL settings written into EMR1.

Forcing DDR2 Commands

The DDR2C has some specific bits which can be used to aid in debug and in specific system solutions.

Force Precharge All

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

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

Force Load Mode Register

Programs can use the Force LMR command by setting bit 22 (=1) in the `DDR2CTL0` register. The Force LMR bit allows changes to the `MODE` register based settings during runtime. These settings include bit 22 (=1) for MR command (settings `DDR2CTL2` register).

Force Auto-Refresh

Bit 20 (=1) in the `DDR2CTL0` register forces the auto refresh to be immediately executed (not waiting until the refresh counter has expired). This is useful for test purposes but also to synchronize the refresh time base with a system relevant time base.

Force Extended Mode Register 1–3

Programs use the Force extended mode register 1–3 commands (`DDR2CTL0` register) by setting:

- bit 23 (=1) for EMR1 command (settings `DDR2CTL3` register)
- bit 12 (=1) for EMR2 command (settings `DDR2CTL4` register)
- bit 17 (=1) for EMR3 command (settings `DDR2CTL5` register)

This allows programs to initialize or change the content of the `EMR` register.

Force DLL External Bank Calibration

The last step during power up is the post calibration of the external DDR2 banks. This command is enabled by setting bit 13 (=1) in the `DDR2CTL0` register. If enabled the DDR2 controller posts 300 dummy reads for calibration between the internal DDR2 clock and the `DDR2_DQS1-0` pins which are driven during the read. Note the calibration is done separately for each assigned external bank.

Data Transfer

The AMI can access data from both the core and through DMA. The following sections describe these options.

Data Buffers

The asynchronous memory interface has two 1 deep data buffers, one each for the transmit and receive operations. These are described in the sections that follow.

AMI Receive Buffer

Reads from external memory are done through the 1 deep receive packing buffer (AMIRX). When an external address that is mapped to the AMI in the EPCTL register is accessed, it receives 8/16-bit data and packs the data based on the packing and control modes in the AMI control register (AMICTLx). Once a full packed word is received, the internal status signal is deasserted and new reads are allowed.

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

AMI Transmit Buffer

Writes to external memory are done through the 1 deep transmit packing buffer (AMITX). When an external address that is mapped to the AMI in the EPCTL register is accessed, it receives data from internal memory using the DMA controller or through direct core writes.

Once a full word is transferred out of the AMI, the internal status signal is deasserted and new writes are allowed. No more external transfers can start while the AMI module is not empty.

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



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

DMA Buffer

The external port supports two DMA channels, each populated with a data buffer (`DFEP1-0`). Each data buffer is 6 locations deep and its status can be read in the `DMACx` register. Note the DMA channels are valid for AMI, SDRAM or DDR2 transfers. [For more information, see “External Port DMA” on page 3-100.](#)

Core Access

For core-driven external port transfers, the instruction needs to read or write from a valid external port address.

External Port Dual Data Fetch

The dual data fetch instruction (Type 1) allows the processor to access external data from both DAGs. In such an instruction, the accesses are executed sequentially (not simultaneously as in internal memory). For example:

```
r4=r2+r3, r2=dm(i6,m6), r3=pm(i10,m10);
```

The DAG1 access (operand `r2`) is executed first followed by the second DAG2 access (operand `r3`).

Data Transfer

Conditional Instructions

On the SHARC processors, almost all instruction types can be conditional. Access to external data based on a conditional instructions are allowed. For example:

```
r10=pass r9;  
If EQ r4=r2+r3, r2=dm(i6,m6);
```

The instruction is only executed if the condition is true.

SIMD Access

The SHARC processor supports SIMD data access from external DDR2 memory. In SIMD mode, the core expects 64-bit data on a single read request and drives 64-bit data for write requests. The controller decodes the access request and if it is a SIMD read from a location N, the controller fetches data from N and N+1, irrespective of whether N is an odd or an even address.

The memory controller then packs the data into 64 bits and sends it back along the core buses. For a SIMD write, the controller unpacks the 64-bit data given by the core and writes it to N and N+1 memory locations. For a SIMD read, the controller reads the physical data from N and N+1 memory locations and packs the 64-bit data to N and N+1 memory locations.

The behavior of SIMD access to/from external memory is similar to the internal processor memory. The only difference is that it is supported in normal word (32-bit) address space only. Unlike internal memory access, SIMD access from external memory may have a different latency, the explicit transfer terminate first followed by the implicit transfer.

SDRAM

SIMD Mode transfers can be performed within 2 core access.

The first access does the explicit 32-bit access (which results in the physical space in 2x16-bit words) while the 2nd 32-bit access does the implicit transfer. In total there are four read or write commands as shown in [Table 3-16](#).

Table 3-16. SDRAM SIMD Access

Access	Logical x32	Physical x16	Comment
Explicit	DM(0x200000) = R0	DM(0x400000) = R0(LSW); DM(0x400001) = R0(MSW);	No Masking
Implicit	DM(0x200001) = S0	DM(0x400002) = S0(LSW); DM(0x400003) = S0(MSW);	No Masking

DDR2

For a SIMD transfer, the controller can burst the 64-bit data given by the core and writes it to N and N+1 memory locations. Since the burst length is 4, SIMD mode transfers can be performed within one burst access.

The first access does the explicit 32-bit access (which results in the physical space in 2x16-bit words) while the 2nd 32-bit access performs the implicit transfer. In total there is one read or write command per burst of 4x16-bit data as shown in [Table 3-17](#).

Table 3-17. DDR2 SIMD Burst Access

Access	Logical x32	Physical x16	Comment
Explicit	DM(0x200000) = R0	DM(0x400000) = R0(LSW); DM(0x400001) = R0(MSW);	DDR2_DM1-0 = low DDR2_DM1-0 = low
Implicit	DM(0x200001) = S0	DM(0x400002) = S0(LSW); DM(0x400003) = S0(MSW);	DDR2_DM1-0 = low DDR2_DM1-0 = low

Data Transfer

-  Bursts are not divisible. During reads, all DDR2 data are received and on-chip masked by the DDR2 controller. For single write access in SISD mode, the 3rd and 4th data needs to be masked. The data masking (DDR2_DM1-0 signal) is only performed during write operations as shown in [Table 3-18](#).

Table 3-18. DDR2 SISD Access

Access	Logical x32	Physical x16	Comment
Explicit only	DM(0x200000) = R0; DM(0x400000) = R0;(LSW) DM(0x400001) = R0;(MSW) DM(0x400002) = S0;(LSW) DM(0x400003) = S0;(MSW)	DDR2_DM1-0 = low DDR2_DM1-0 = low DDR2_DM1-0 = high DDR2_DM1-0 = high	

-  SIMD write access to the DDR2 memory should be even address aligned. If odd address aligned, the throughput is reduced by a factor of 2. This does not apply to SIMD reads or any SISD mode. For more information on SIMD access, see the *SHARC Processor Programming Reference*.

External Instruction Fetch

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

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

Interrupt Vector Table (IVT)

The interrupt vector table can be located in the internal ROM (0x80 000, $IIVT$ bit = 0) or internal RAM (0x8C 000, $IIVT$ bit = 1) based on the

selected boot mode. However for all boot modes except the reserved boot mode, the default IIVT bit setting is 1 (SYSCTL).

Therefore, if instruction fetch from external memory is desired 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 fetch these instructions.

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

Fetching ISA Instructions From External Memory

The SDRAM/DDR2 controllers along with the processor core incorporates appropriate enhancements so that instruction code can be fetched from the SDRAM/DDR2 at the maximum possible throughput.

Throughput is limited only by the SDRAM/DDR2 when the code is non sequential.

The address map for code is same as for data. Each address refers to a 32-bit word. Any address produced by the sequencer is checked to determine if it falls in the external memory and if so, the SDRAM/DDR2 controllers initiate access to the SDRAM/DDR2. Because the sequencer address bus is limited to 24 bits, only part of the external memory address area can be used to store code. As explained in the following section, the address generated by the sequencer undergoes translation to produce a physical address, since the SDRAM data bus width is less than 48 bits.



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

Data Transfer

Instruction Packing

Any address produced by the sequencer which falls in external memory is first translated into the physical address in external memory based on the actual data bus width of external memory as shown in [Figure 3-18](#).

The controller completes the required number of accesses from consecutive locations for returning a 48-bit word instructions. For a 16-bit SDRAM/DDR2 bus, it performs three accesses.



Only bank0 can be populated for external instruction fetch.

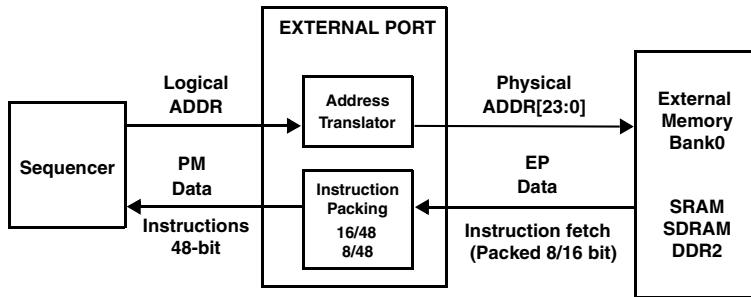


Figure 3-18. Logical Versus Physical Addresses

16-Bit Instruction Storage and Packing

In [Table 3-19](#) the logical to physical translation is a multiplication by a factor of 3 and $N = 0x355554$. Therefore, the 16-bit wide AMI memory supports 3.3 million instructions.

In [Table 3-19](#) $P = 0xE00000$. Therefore, the total number of external memory instructions for a 16-bit wide SDRAM/DDR2 memory is 14 million.

Table 3-19. Logical Versus Physical Address Mapping, 16-Bit AMI

Logical ISA Normal Word Address, Program Sequencer	Physical Address, External Bus	Data15–0
0x20 0000	0x60 0000	Instr0[15:0]
	0x60 0001	Instr0[31:16]
	0x60 0002	Instr0[47:32]
0x20 0001	0x60 0003	Instr1[15:0]
	0x60 0004	Instr1[31:16]
	0x60 0005	Instr1[47:32]
0x20 0002	0x60 0006	Instr2[15:0]
	0x60 0007	Instr2[31:16]
	0x60 0008	Instr2[47:32]
...	...	
0x55 5554 (top AMI address)	0xFF FFFD	InstrN[15:0]
	0xFF FFFE	InstrN[31:16]
	0xFF FFFF	InstrN[47:32]
0x5F FFFF	0x11F FFFD	InstrN[15:0]
	0x11F FFFE	InstrN[31:16]
	0x11F FFFF	InstrN[47:32]

8-Bit Instruction Storage and Packing

In [Table 3-20](#), the logical to physical translation is a multiplication by a factor of 6 and $N = 0xAAAA9$. Therefore, the 8-bit wide AMI supports 0.7 million instructions.

Table 3-20. Logical Versus Physical Address Mapping,
8-Bit AMI

Logical ISA Normal Word Address, Program Sequencer	Physical Address, External Bus	Data7–0
0x20 0000	0xC0 0000	Instr0[7:0]
	0xC0 0001	Instr0[15:8]
	0xC0 0002	Instr0[23:16]
	0xC0 0003	Instr0[31:24]
	0xC0 0004	Instr0[39:32]
	0xC0 0005	Instr0[47:40]
0x20 0001	0xC0 0006	Instr1[7:0]
	0xC0 0007	Instr1[15:8]
	0xC0 0008	Instr1[23:16]
	0xC0 0009	Instr1[31:24]
	0xC0 000A	Instr1[39:32]
	0xC0 000B	Instr1[47:40]
...	...	
0x2A AAA9	0xFF FFFC	InstrN[23:16]
	0xFF FFFD	InstrN[31:24]
	0xFF FFFE	InstrN[39:32]
	0xFF FFFF	InstrN[47:40]

Mixing Instructions and Data in External Bank 0

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

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

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

Addressing for Various Memory Sizes

[Table 3-21](#) provides addressing for various sizes of DDR2 DRAM memory.

Table 3-21. Translation of Logical to Physical Addressing for DDR2

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

Data Transfer

[Table 3-22](#) provides addressing for various sizes of SDRAM memory.

Table 3-22. Translation of Logical to Physical Addressing for SDRAM

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

Writing Instructions to External Memory

As described in the previous sections, the sequencer fetches instructions from physical 16-bit addresses, regardless of whether they are ISA or VISA instructions. However, to boot instructions into external memory via core or DMA, normal word space is required. Since the memory space is aliased the short word address space is accessible in normal word space, which is a right shift of the short word address ($0x60\ 0000 \gg 1$ is $0x30\ 0000$).

As shown in [Table 3-23](#), instructions are stored in an interleaved fashion which ensures that the memory is used efficiently. For every 2 fetches (VISA) or instructions (ISA) 3 memory accesses are required. For more information refer to the Visual DSP tools loader file documentation.

Table 3-23. Booting Instructions Into External Memory

Sequencer Feych Address	Normal Word Address	Normal Word Data	
0x20 0000	0x30 0000	Instr/Fetch 0 [31:0]	
	0x30 0001	Instr/Fetch1 [15:0]	Instr/Fetch0 [47:32]

Table 3-23. Booting Instructions Into External Memory

Sequencer Feych Address	Normal Word Address	Normal Word Data	
0x20 0001	0x30 0002	Instr/Fetch1 [47:16]	
	0x30 0003	Instr/Fetch2 [31:0]	
0x20 0002	0x30 0004	Instr/Fetch3 [15:0]	Instr/Fetch2 [47:32]
	0x30 0005	Instr/Fetch3 [47:16]	

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 3-19](#) shows the format for storing an instruction.

Data Transfer

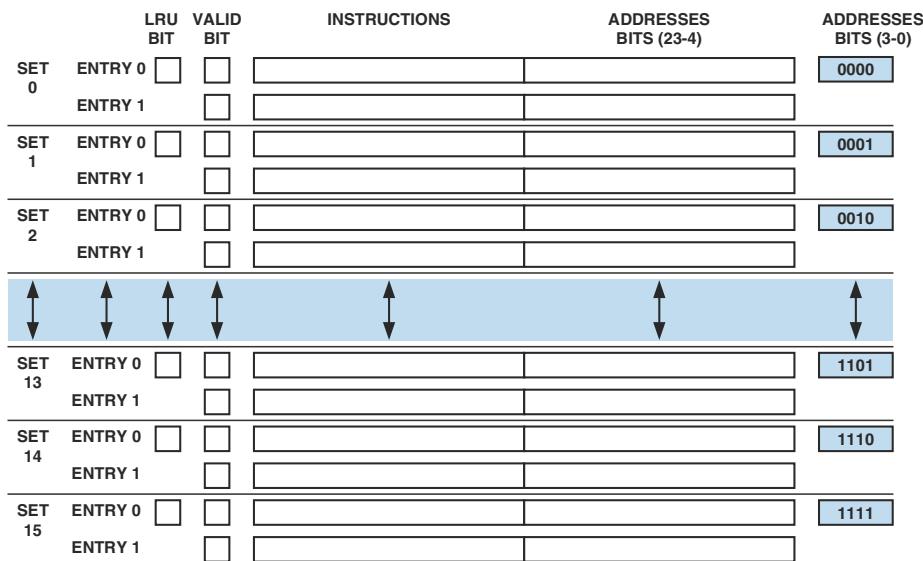


Figure 3-19. Instruction Cache Architecture

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

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

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

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

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

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

```
lctr=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 (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 does not play a significant role in improving the efficiency of strictly linearly executed code from external memory.

Fetching VISA Instructions From External Memory

The SHARC processors support fetching instructions from external SDRAM or DDR2 memory. These instructions may be stored either as traditional 48-bit SHARC ISA instructions, or as VISA instructions.

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

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

Just as the same physical internal memory on the processors can be accessed and addressed in many different ways, the external memory space can also be viewed either as logical or physical addresses. To support VISA

in external memory, the external memory address range has been divided into two ranges:

- Normal word – 0x20 0000 to 0x5F FFFF
- Short word – 0x60 0000 to 0xFF FFFF

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

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

Note that that external port interface passes the addresses in the range 0x60 0000 – 0xFF FFFF as is to external memory. As in the previous case, the external port accesses three short words to return a 48-bit word to the IAB for each access requested by the sequencer. The short words for a VISA section of code are packed in such a way that lowest of the addresses pertaining to a given instruction has the most significant short word of that instruction and the highest address has the least significant short word (see [Figure 3-20](#)).

External Port DMA

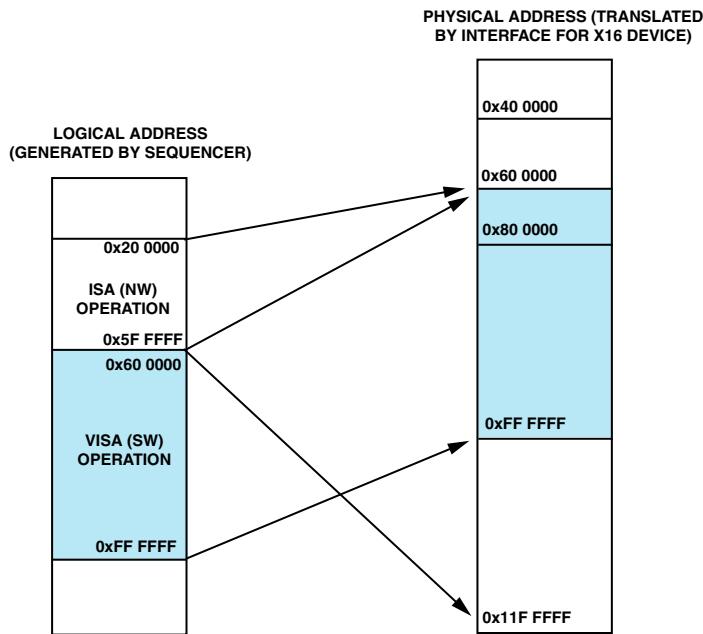


Figure 3-20. Translation of Logical to Physical External Memory Addressing

External Port DMA

The external port has two DMA channels that can use either the SDRAM/DDR2 controller or the asynchronous memory interface (AMI). The AMI controller supports DMA with an external data width of 8 bits. The SDRAM/DDR2 controllers support DMA with an external data width of 16-bits.

External Port DMA Parameter Registers

These registers are used to set up and control DMA through the processor's external port. For information on these registers and on how to set

up DMA transfers, see “[General Procedure for Configuring DMA](#)” on page [2-51](#). The registers that control external port DMA are described [Table 3-24](#).

Table 3-24. DMA Parameter Registers

Register	Description	Comment
IIEPx	Internal Index	Internal Start Address. For delay line DMA, it serves as the delay line write index; for example, the start address of the internal memory buffer for the external write data.
IMEPx	Internal Modifier	Internal address modifier.
ICEPx	Internal Count	For delay line DMA, it serves as count for delay line writes, write block size.
EIEPx	External Index	External start address.
EMEPx	External Modifier	External address modifier.
ECEPx	External Count	External memory count, read only (alias of ICEPx)
CPEPx	Chain Pointer	Contains address of the next descriptor in internal memory.

Table 3-25. Enhanced DMA Parameter Registers

Register	Description	Comment
ELEPx	Circular Buffer Length	Hold circular buffer length for circular, delay line DMA, scatter/gather DMA.
EBEPx	External Base	Hold circular start address for circular, delay line DMA, scatter/gather DMA.
RIEPx	Read Internal Index	Contains start address of internal memory buffer to which the data read from external memory during delay line DMA reads are to be written into (alias of IIEPx during delay line DMA).
RCEPx	Read Count	Contains number of reads from each taplist, read block size (alias of ICEPx during delay line DMA).
RMEPx	Read External Modifier	Contains external modifier to be used for delay line reads (alias of EMEPx during delay line DMA).

External Port DMA

Table 3-25. Enhanced DMA Parameter Registers (Cont'd)

Register	Description	Comment
TCEPx	Tap Count	Holds the length of the tap list (the number of taps for delay line DMA, scatter/gather DMA).
TPEPx	Tap List Pointer	Holds address of an array in internal memory which holds offsets to be used when accessing delay line DMA in external memory. The offset represents the first address of each read block. Applies to delay line DMA, scatter/gather DMA

Operating Modes

This section highlights the different DMA modes which can be used with the external port.

Internal DMA Addressing

Besides the traditional internal to external addressing type, the DMA module also supports internal to internal transfers. This is accomplished by indexing all external parameter registers with internal addresses. The DMA controller recognizes the transfer by addresses and not by an additional control bit setting.



Note that the DMA channel priority changes if using internal vs. external index addresses.

The SHARC supports another internal to internal DMA module (MTM) which has higher priority by default but does only support standard DMA mode. For more information, see [Chapter 5, “Memory-to-Memory Port DMA”](#).

Standard DMA

This DMA type resembles the traditional DMA type to initialize the different internal and external parameters (index, modify and count) registers and configuration of the DMA control registers.

Note that the ECEP parameter register (read only) is a copy of the ICEP register. If ICEP is written, the ECEP register is updated automatically ([Figure 3-21](#)).

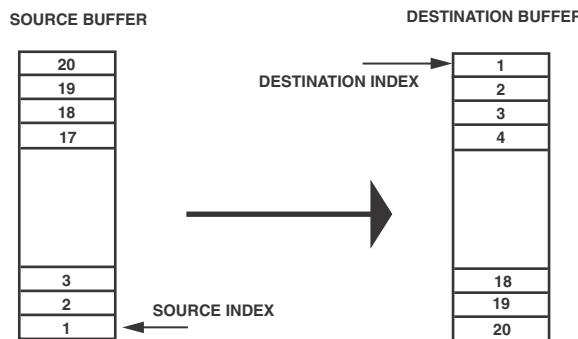


Figure 3-21. Standard Write

Circular Buffered DMA

Circular buffered DMA ([Figure 3-22](#), [Figure 3-23](#)) resembles the traditional core DAG circular buffered mode by using registers for circular buffering. In this mode the DMA needs two additional registers (base and length) to support reads and writes to a circular buffer.

External Port DMA

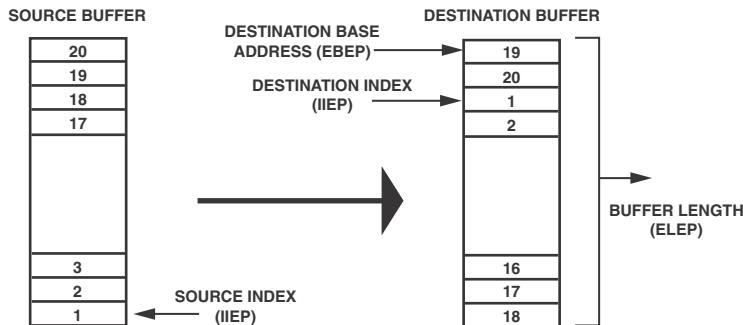


Figure 3-22. Circular Buffering Write DMA

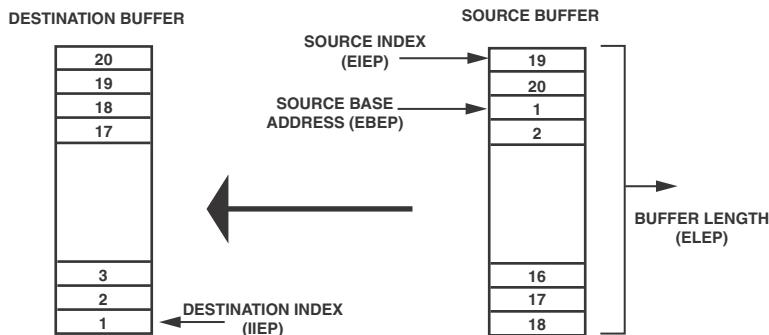


Figure 3-23. Circular Buffering Read DMA

Chained DMA Mode

Chained DMA is used to support repetitive reads and writes to a specific area which is defined by the individual TCBs.

Changing DMA Direction on the Fly

The SHARC processors allow a change of external port data direction for each individual TCB in a chain sequence.

Bit 20 (**CPDR**) bit of the external port chain pointer register (**CPEPx**) changes the data flow direction. If **CPDR** is cleared (=0) writes to internal memory are performed, if **CPDR** is set (=1), internal memory reads are performed. This works similar to the **PCI** bit (bit 19). Bit 8 (**OFCEN**) and bit 2 (**CHEN**) in the **DMACx** register must be set (=1) to enable this functionality.

Listing 3-7. Changing DMA Direction

```
.section/pm seg_dmdu;
/* EP TCB storage order CP-EM-EI-C-IM-II */
.var TCB1[6] = 0 , M , extbuffer , N , M , buffer;
.var TCB2[6] = 0 , M , extbuffer , N , M , buffer;

.section/pm seg_pmco;
R0=0;
dm(CPEP0)=R0;           /* clear CPx register */

r0 = DEN|CHEN|OFCEN;      /* enable DMA channel */
dm(DMAC0)=r0;

R2=(TCB1+5) & 0xFFFF;    /* load IIx address of next TCB and
                           mask address */
R2=bset R2 by 19;         /* set PCI bit */
dm(TCB2)=R2;             /* write address to CPx location of
                           current TCB */
R2=(TCB2+5) & 0xFFFF;    /* load IIx address of next TCB and
                           mask address*/
R2=bset R2 by 19;         /* clear PCI bit */
R2=bset R2 by 20;         /* set CPDR bit */
dm(TCB1)=R2;             /* write address to CPx location of
                           current TCB */
dm(CPEP0)=R2;             /* write IIx address of TCB1 to CPx
                           register to start chaining*/
```

External Port DMA



If chaining is enabled with the OFCEN bit set then the TRAN bit has no effect, and direction is determined by the CPDR bit in the CPEP register.

Scatter/Gather DMA

The purpose of scatter/gather DMA ([Table 3-26](#), [Figure 3-24](#) through [Figure 3-27](#)) is the transfer of data from/to non contiguous memory blocks.

The scatter/gather DMA type is a fixed block size scatter/gather DMA that relies on tap list entries in internal memory to calculate the external address to scatter/gather the DMA. If the DMA direction is external write (TRAN = 1) then it is a scatter DMA. If TRAN = 0 then it is a gather DMA. This mode also supports chained and circular buffer chained DMAs.

Table 3-26. External Read/Write Index Calculation
Scatter/Gather DMA

Equation	Result
EIEP + TL[N]	First address for tap N
EIEP + TL[N] + 1 × EMEP	Second address for tap N
EIEP + TL[N] + 2 × EMEP	Third address for tap N
...	
EIEP + TL[N] + ICEP × EMEP	Final address for tap N
EIEP + TL[N + 1]	First address for tap N + 1
EIEP + TL[N + 1] + 1 × EMEP	Second address for tap N + 1

External Address Calculation

For scatter/gather DMA, the tap list modifiers are employed and the number of taps is determined by the tap list count register (TCEPx). The number of sequential reads (block size) from every tap is determined by

the internal count register (`ICEPx`), and is the same for every tap. The read/write pointer in external index register (`EIEPx`) serves as the index address for these read/writes.

`TL[N]` is the first tap list entry in the internal memory as pointed by the `TPEP`, the tap list pointer. The tap list entries are 27-bit signed integers. Therefore, for each read/write block, the DMA state machine fetches the offset from the tap list. The offset is added to the `EIEP` value to get the start address of the next block. The external addresses are circular buffered if circular buffering is enabled ([Figure 3-26](#), [Figure 3-27](#)).

Once the `ICEP` register for the final tap decrements to zero (both `TCEP` and `ICEP` are zero), then the tap list DMA access is complete and the DMA completion interrupt is generated (if chaining is enabled the interrupt depends on the `PCI` bit setting).

The write back mode (`WRBEN` bit) is not applicable for tap list based DMA (as the addressing is pre-modify, and therefore the `EIEP` value coincides with the `TCB` value even at the end of DMA). So even if the `WRBEN` bit is set in tap list DMA mode, the write backs do not occur.

External Port DMA

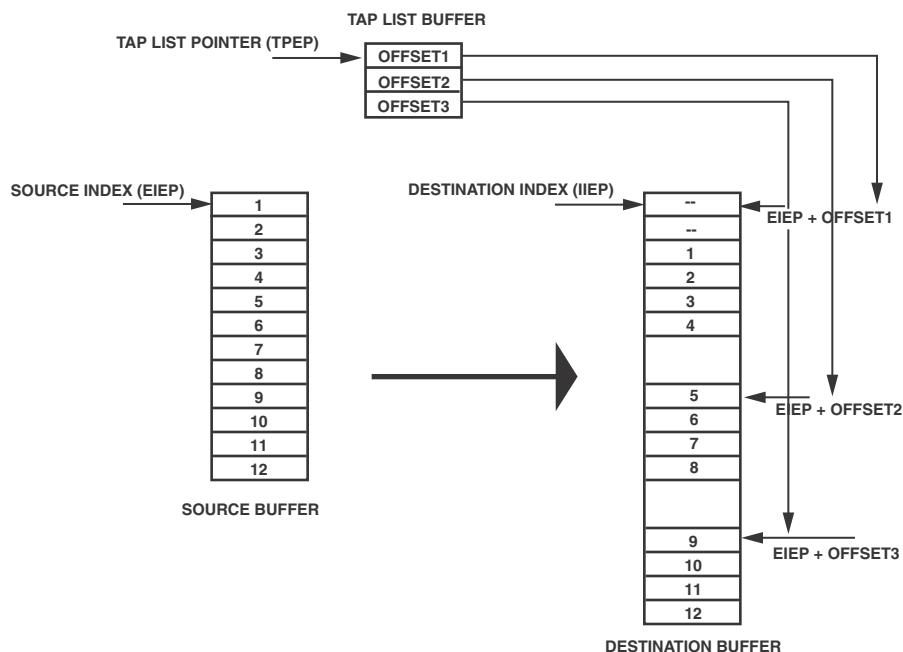


Figure 3-24. Scatter DMA (Writes)

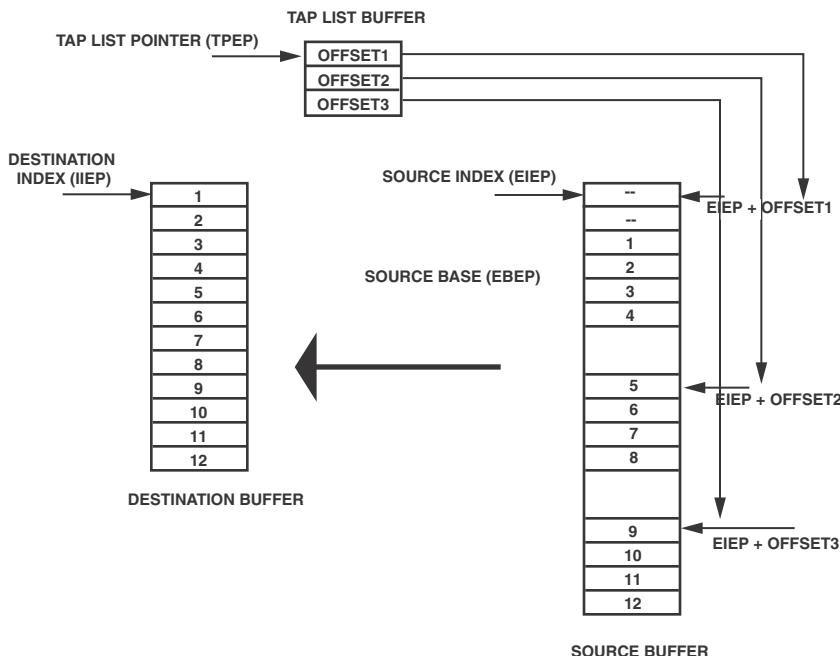


Figure 3-25. Gather DMA (Reads)

External Port DMA

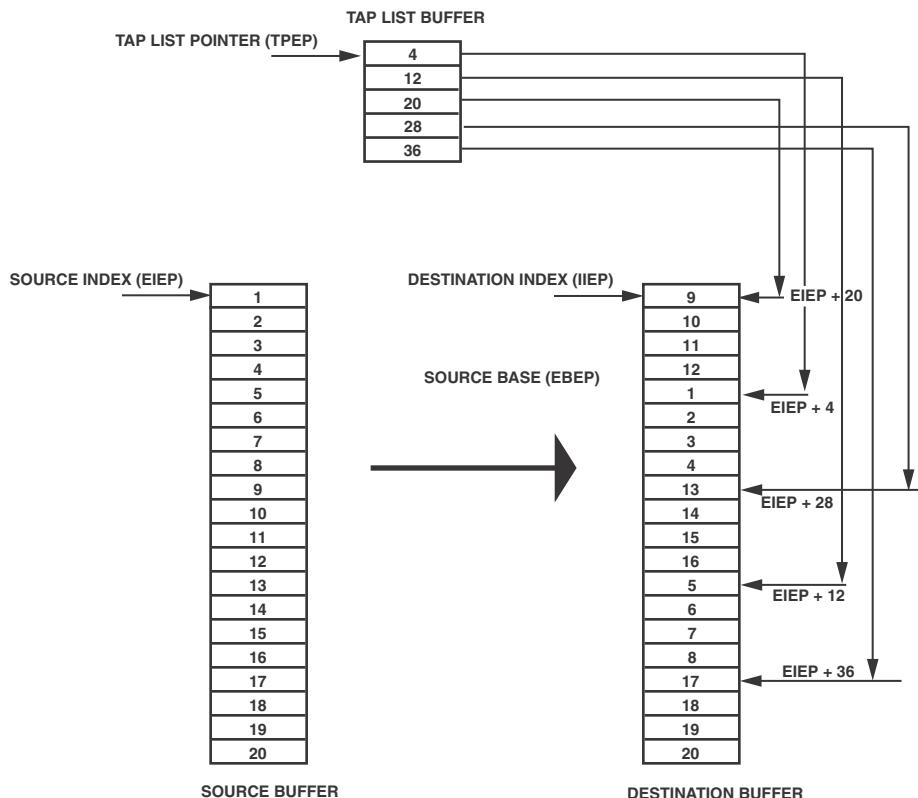


Figure 3-26. Circular Buffering Scatter DMA (Writes)

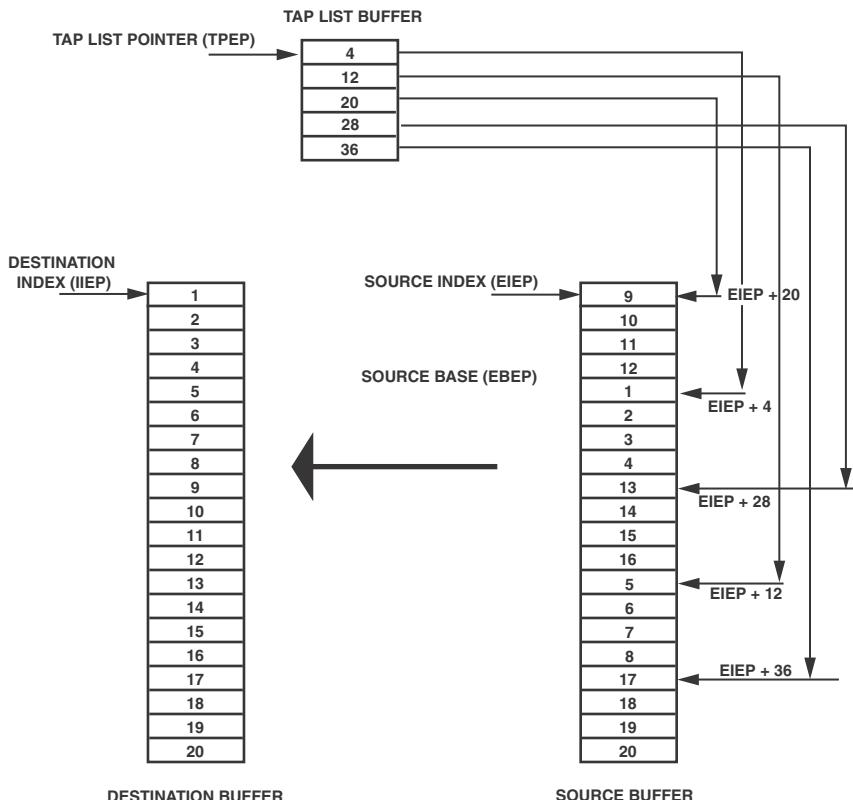


Figure 3-27. Circular Gather DMA (Reads)

Delay Line DMA

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

The delay line DMA access consists of the following accesses in the order listed and is shown in [Figure 3-28 on page 3-113](#). Note that in the figure

External Port DMA

single reads from each TAP are shown for simplicity and block reads are default, depending on the count specified in the `RCEP` register.

1. Writes to external memory. The number of writes is determined by the `ICEP` register. The data is fetched from the `IIEP` register and the `IMEP` register is used as the internal modifier. The `EIEP` register serves as the external index and is incremented by the `EMEP` register after each write. These writes are circular buffered if circular buffering is enabled.
2. In chained DMA, when the writes are complete, (`ICEP` = zero) the `EIEP` 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 memories. For reads, the tap list (TL) modifiers are used and the number of reads is determined by the `RCEP` register. The write pointer in the `EIEP` register 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. Therefore, for each read, the DMA controller fetches the external modifier from the tap list and the reads are circular buffered (if enabled).

External Address Calculation for Reads

Note that `TL[N]` is the first tap list entry in internal memory pointed to by the tap list pointer register (`TPEP`). Tap list entries are 27-bit signed integers. Therefore, for each read-block, the DMA state machine fetches the offset external modifier from the tap list. The reads are circular buffered if circular buffering is enabled.



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

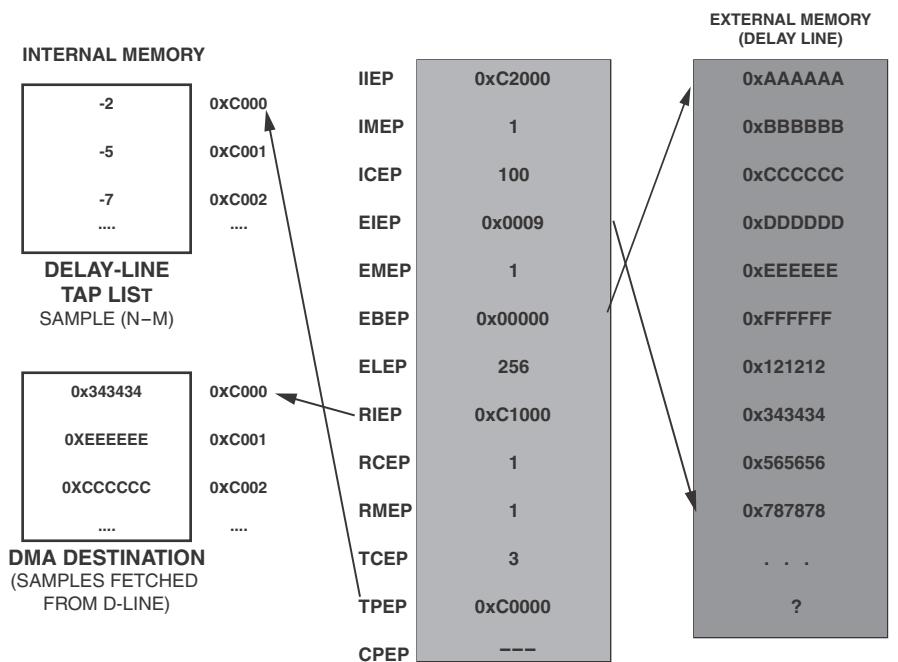


Figure 3-28. Delay Line DMA Reads

these delay line reads. Once the read count completes, the `ICEP` register decrements to zero (both `ICEP` and `TCEP` are zero) for the final tap. Finally, the delay line DMA access completes and the DMA completion interrupt is generated. If chaining is enabled, the interrupt is dependent on the `PCI` bit setting. The delay line DMA can only be initialized using the TCB. In order to use the delay line DMA for a single DMA sequence, initialize the `CPEP` register to zero in the TCB.

For each 32-bit tap read, the external read index is shown in [Table 3-27](#). Note that one tap list entry starts multiple reads.

Interrupts

Table 3-27. External Read Index Calculation Delay Line DMA

Equation	Result
EIEP + TL[N]	First read address for tap N
EIEP + TL[N] + 1 × RMEP	Second read address for tap N
EIEP + TL[N] + 2 × RMEP	Third read address for tap N
...	
EIEP + TL[N] + RCEP × RMEP	Last read address for tap N
EIEP + TL[N + 1]	First read address for tap N + 1
EIEP + TL[N + 1] + 1 × RMEP	Second read address for tap N + 1

Interrupts

There are two external port DMA channels. The following sections describe the two ways of triggering interrupts. [Table 3-28](#) provides an overview of external port interrupts.

Table 3-28. External Port Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
External port DMA (2 channels)	DMA RX/TX complete	–internal transfer completion –access completion	RTI instruction	P9I, P13I

Access Completion

This is the default mode of interrupt generation where the DMA complete interrupt is generated when accesses are completed.

- For external write DMA, the DMA complete interrupt is generated only after external writes on the DMA external interface are done.
- For external read DMA, the DMA complete interrupt is generated when the internal DMA writes complete.

In this mode, the DMA interface can be disabled as soon as the interrupt is received, (there is no need to check `EXTS` before disabling the DMA interface).



The DMA interface can be disabled based on a DMA complete interrupt. However, the external device interfaces—AMI/SDRAM may still be performing writes of the DMA data. Prior to disabling any of these devices, programs should check their respective status bits.

Internal Transfer Completion

This mode of interrupt generation is enabled when `INTIRT` bit is set in the DMA control register. This mode of interrupt generation resembles traditional SHARC DMA interrupt generation and is provided for backward compatibility. This interrupt is generated once the DMA internal transfers (transmit or receive) are completed. For external transmit DMA, there may be still external access pending at the external DMA interface when the completion interrupt is generated. Therefore, the DMA may be disabled on the DMA complete interrupt only if the external interface is idle (for example, `EXTS` = 0).

Interrupt Dependency on DMA Mode

Interrupt generation varies, depending on the DMA mode used. The `INTIRT` bit determines whether the interrupt is generated on internal

External Port Throughput

completion or access completion. The following also effect interrupt generation.

- For standard chained DMA, if the `PCI` bit is cleared (= 0), the DMA complete interrupt is generated only after the entire chained DMA access is complete. If the `PCI` bit is set (= 1), then a DMA interrupt is generated for each TCB.
- For the delay line DMA, the DMA complete interrupt is generated when both the write access and the delay line reads are completed. In a chained delay line DMA, the `PCI` bit determines if each delay line TCB generates an interrupt or not.
- With scatter/gather DMA, the DMA complete interrupt is generated only after all tap list reads/writes are complete. As in the delay line DMA, the `PCI` bit setting determines if each tap list TCB generates an interrupt in a chained access.



If DMA is disabled in the middle of data transfers, the DMA interrupts should not be used.

External Port Throughput

The following sections provide information on the throughput of the external port interfaces (AMI, SDRAM).

AMI Data Throughput

The AMI data throughput is shown in [Table 3-29](#).

Table 3-29. Read/Write Throughput

Access ¹	8-Bit I/O	16-Bit I/O
Write	32-bit word per 12 cycles	32-bit word per 6 cycles
Read	32-bit word per 12 cycles	32-bit word per 6 cycles

1 Throughput for minimum wait states of 2 with no idle and hold cycles.

SDRAM Throughput

[Table 3-30](#) provides information needed to configure the SDRAM interface for the desired throughput.

Table 3-30. SDRAM Data Throughput

Access	Page	Throughput per SDCLK (16-Bit Data)
Sequential uninterrupted reads	Same	One word per two cycles ¹
Any writes	Same	2 cycles
Non sequential uninterrupted reads	Same	7 cycles

1 Optimization enabled, first data of a sequential read takes 7 cycles for CL = 2 and 8 cycles for CL = 3, thereafter it is one word per two cycles.

Throughput Conditional Instructions

A conditional read/write may take 1 PCLK cycle (access made and access aborted, respectively). [For more information, see “External Memory Access Restrictions” on page 3-132](#).

External Port Throughput

DDR2 Throughput

The following sections provide information needed to configure the DDR2 interface for the desired throughput.

DMA Throughput

[Table 3-31](#) provides approximate throughput information with the processor core running at 400 MHz for DMA-driven reads and writes of external DDR2 memory. The throughput numbers shown are measured by running chained DMA with four TCBs (with 256 32-bit words per transfer block).

For the analysis, 16 bit DDR2 is used ($t_{FAW}=10$, $t_{RRD}=2$, $t_{RTP}=2$, $t_{MRD}=3$, $t_{RCRD}=3$, $t_{TWTR}=1$, $t_{RP}=3$, $t_{RAS}=8$, $CL=4$, $AL=4$, $t_{WR}=4$).

Throughput is calculated by measuring time between the instant when DMA is enabled and instant when DMA completion ISR is entered.

Table 3-31. DMA Throughput, 400 MHz Core Clock

Operation	DDR2 Clock	Clock Ratio	Throughput
DMA Reads			
	133 MHz	1:3	473M bytes/sec.
	200 MHz	1:2	700M bytes/sec.
DMA Writes			
	133 MHz	1:3	481M bytes/sec.
	200 MHz	1:2	732M bytes/sec.

Core Throughput

[Table 3-32](#) provides approximate throughput information with the processor core running at 400 MHz for core-driven reads and writes of

external DDR2 memory. The throughput numbers shown are measured by running a loop of 1024 read/writes (512 in case of SIMD reads/writes).

For the analysis, 16-bit DDR2 is used ($t_{FAW}=10$, $t_{RRD}=2$, $t_{RTP}=2$, $t_{MRD}=3$, $t_{RCD}=3$, $t_{WTR}=1$, $t_{RP}=3$, $t_{RAS}=8$, $CL=4$, $AL=4$, $t_{WR}=4$).

Throughput is calculated from start of the first iteration of the loop to the end of the last iteration of the loop.

Table 3-32. Core Throughput, 400 MHz Core Clock

Operation	DDR2 Clock	Clock Ratio	Throughput
Core Reads (SISD/SIMD)			
	133 MHz	1:3	495M bytes/sec.
	200 MHz	1:2	742M bytes/sec.
Core Writes (SISD)			
	133 MHz	1:3	529M bytes/sec.
	200 MHz	1:2	793M bytes/sec.
Core Writes (SIMD)			
	133 MHz	1:3	531M bytes/sec.
	200 MHz	1:2	796M bytes/sec.

External Instruction Fetch Throughput

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 throughput is maximized when SDRAM is programmed to run at half the core clock frequency and the instructions being fetched are sequential.



Read optimization logic does not apply to external code execution.

Effect Latency

Table 3-33 illustrates the performance of code execution depending on different access types.

Table 3-33. Core Throughput

Access	Data Width	Page	Throughput per SDCLK (CL = 2)	Throughput per SDCLK (CL = 3)
Sequential uninterrupted reads	16	Same	2 instructions per 6 cycles*	2 instructions per 6 cycles*
Non sequential uninterrupted reads	16	Same	1 instructions per 9 cycles	1 instructions per 10 cycles

* In this case SDC has to fetch 3 data (16-bit) for each instruction.
First 48-bit instruction of a sequential read will take 8 cycles for CL = 2 and 9 cycles for CL = 3, thereafter it is two instructions per 6 cycles.
The instruction available cycles will look like - 8, 10, 14, 16, 20, 22, 26, 28 ... (CL = 2)

When executing from external asynchronous memory, instruction throughput depends on the settings of asynchronous memory such as the number of wait states, the ratio of core to peripheral clock and other settings. For details, please refer to the external port global control register (EPCTL), the AMICTL_x register, and the SDCTL0 register in “[ADSP-2147x, ADSP-2148x External Port Registers](#)” on page [A-45](#).

Effect Latency

The total effect latency is a combination of the write effect latency (core access) plus the peripheral effect latency (peripheral specific). After the AMI/SDRAM/DDR2 registers are configured the effect latency is 1.5 PCLK cycles minimum and 2 PCLK cycles maximum.

After the external port register is configured the effect latency is 4 PCLK cycles. This is the valid for the worst case of core to SDRAM/DDR2 clock ratio of 1:4

Write Effect Latency

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

Programming Models

The following sections provide information on the various programming models that are used through the external port interface.

External Port

The section describes software programming steps required for the successful operation of the external port.

DMA

The following sections describe the programming steps for different types of DMA transfers.

Standard DMA

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

1. Configure the `AMICTLx` registers to enable the AMI and to set the desired wait states, data bus width, and so on. Configure the `SDCTL` registers to enable SDRAM/DDR2, and to set the desired clock and timing settings, the data bus width, and other parameters.
2. Initialize the `IIEP`, `IMEP`, `ICEP`, `EIEP`, and `EMEP` registers.
3. If circular buffering is desired, use the corresponding TCB storage.

Programming Models

4. If scatter/gather DMA is desired, program additional writes to the `TCEP` and `TPEP` registers.
5. Enable DMA using the `DMAEN` bit, and set the transfer direction using the `TRAN` bit in the `DMACx` registers. If scatter/gather DMA is desired, set the `TLEN` bit. It is advised that the DMA FIFOs are flushed using the `DFLSH` bit when DMA is enabled.

Once the DMA control register is initialized, the DMA engine fetches the DMA descriptors from the address pointed to by `CPEP`. Once the DMA descriptors are fetched then the DMA (or the tap list DMA) process starts. Once the DMA (or tap list DMA) is complete, the new DMA descriptors are loaded and the process is repeated until `CPEP` = 0x0. A DMA completion interrupt is generated at the end of each DMA block or at the end of entire chained DMA, depending on the `PCI` bit setting.

Chained DMA

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

1. Clear the chain pointer register.
2. Configure the `AMICTLx` registers to enable the AMI, set the desired wait states, the data bus width, and so on. Configure the `SDCTL` register to enable the SDRAM/DDR2, configure the desired clock and timing settings, data bus width, and other parameters.
3. Initialize the `CPEP` register and set the `PCI` bit if interrupts are required after the end of each DMA block. Set the `CPDR` bit if different DMA direction is required in conjunction with the `OFCEN` bit in the `DMACx` register.

4. If circular buffering is needed, use the corresponding TCB storage.
5. Enable DMA using the `DMAEN`, bit, set chaining using the `CHEN` bit. If circular buffering is required, set the `CBEN` bit in the `DMACx` registers. It is advised that programs flush the DMA FIFOs using the `DFLSH` bit when DMA is enabled.

Once the DMA control register is initialized, the DMA controller fetches the DMA descriptors from the address pointed to by the external port chain pointer register (`CPEP`).

Once the DMA descriptors are fetched, the normal DMA process starts. Upon completion, new DMA descriptors are loaded and the process is repeated until `CPEP` = 0x0. A DMA completion interrupt is generated at the end of each DMA block or at the end of an entire chained DMA, depending on the `PCI` bit setting.

Delay Line DMA

1. Configure the `AMICTLx` register with the desired wait states, enable AMI, data bus width and other parameters.
2. Initialize the `CPEP` register and set the `PCI` bit if interrupts are required after the end of each delay line DMA block.
3. Enable DMA (`DMAEN`), delay line DMA (`DLEN`), chaining (`CHEN`) if required in the `DMACx` register. Programs should flush the DMA FIFO (`DFLSH`) along with enabling the DMA. If circular buffering is required (which is normally the case) enable it by setting the `CBEN` bit.

Once the DMA control register is initialized the DMA engine fetches the DMA descriptors from the address pointed to by the `CPEP` register. Once the delay line DMA access is complete, the new DMA descriptors are loaded and the process is repeated until `CPEP` = 0x0. A DMA completion

Programming Models

interrupt is generated at the end of each delay line DMA block or at the end of entire chained DMA, depending on the `PCI` bit setting.

-  When delay line DMA is enabled with chaining, all the chained DMA blocks follow the delay line DMA access procedure. It is not possible to mix normal DMA with delay line DMA in chained DMA.

Disabling and Re-enabling DMA

Use the following programming model to disable the external port DMA during transfers.

1. Clear the `DMAEN` bit on the `DMACx` register.
2. Wait until the `EXTS` bit is 0.
3. Write 0x0 to the `ICEP` and `DMACx` registers. In cases where DMA is used without chaining, writing to `ICEP` is not required.
4. Re-initialize the required DMA registers, and enable the `DMACx` register while flushing the data/tap list FIFO.

Additional Information

1. If DMA is disabled in the middle of a data transfer, then DMA interrupts cannot be relied on.
2. A single DMA (no chaining) can be stopped midway by clearing the `DMAEN` bit in the `DMACx` register and then restarted from the point where it was stopped by re-enabling the `DMAEN` bit. This mode of inhibiting the DMA only works with single DMA. If a chained/delay line DMA is disabled by clearing `DMAEN` bit then the DMA should be reprogrammed again following the above programming model.

3. For a chained DMA, new TCB loading can be inhibited by clearing the CHEN bit while keeping all other control bits the same. The new TCB is loaded once CHEN bit is re-enabled. The TCB load which was happening when CHEN was cleared will complete.
4. Before initializing a chained DMA (including delay line) make sure that the ICEP and ECEP registers are zero.
5. The DMA parameter registers (except DMACx) should not be written to while chaining is occurring (the CHS bit is set), but any register can be read during chaining.
6. A zero count for the ICEP, RCEP and TCEP registers is forbidden. If a chain pointer with such a descriptor is programmed then the DMA might hang. So a read count zero or a write count zero for a delay line DMA is also forbidden.

AMI Initialization

After reset, the SDCLK is running with the default PLL settings. However, the AMI must be configured and initialized. In order to set up the AMI, use the following procedure. Note that the registers must be programmed in order.

1. Choose a valid CCLK to SDCLK clock ratio in the PMCTL register.
2. Assign external banks to the AMI using the EPCTL register (default).
3. Wait at least 8 PCLK cycles (effect latency).
4. Enable the global AMIEN bit and program the AMI control (AMICTLx) registers. (Define control settings for AMI based on SDCLK speed and asynchronous memory specifications.)

The AMIMS and AMIS bits 1–0 of the AMI status register (AMISTAT) can be checked to determine the current state of the AMI.

SDRAM Controller

This section describes software programming steps required for the successful operation of the SDC.

Power-Up Sequence

After reset, the `SDCLK` is running with the default PLL settings. However, the SDC must be configured and initialized. 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. Choose a valid `CCLK` to `SDCLK` clock ratio in the `PMCTL` register.
2. Wait at least 15 core clock cycles until the new `SDCLK` frequency has been settled up correctly.
3. Assign external banks to SDC in the `EPCTL` register.
4. Wait at least 8 core clock cycles (effect latency).
5. Program the refresh counter in the `SDRRC` register.
6. Define global control for SDC and SDRAM based on speed and SDRAM specifications in the `SDCTL` register.
7. Once the `SDPSS` bit in the `SDCTL` register is set to 1, a dummy access is required to start the power-up sequence.

The SDRAM is ready for access.

The `SDRS` bit (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.

Output Clock Generator Programming Model

The following non VCO programming sequence may be used to change the output generator clock and the core-to-peripheral clock ratio (for example the SDRAM clock). Note that if your program is only changing the PLL output divider, programs do not need to wait 4096 CLKIN cycles (required only if the PLL multiplier or the INDIV bit is modified).

1. Disable the peripheral (SDRAM). Note that the peripherals cannot be enabled when changing clock ratio.
2. Select the PLL divider by setting the `PLLDX` bits (bits 6–7 in the `PMCTL` register).
3. Select the clock divider (`CCLK` to SDRAM ratio) by setting the ratio bits (`PMCTL` register).
4. Wait 15 `CCLK` cycles. During this time, programs must not execute any valid instructions.
5. Enable the peripheral (SDRAM).

The new divisor ratios are picked up on the fly and the clocks smoothly transition to their new values after a maximum of 15 core clock `CCLK` cycles.

Self-Refresh Mode

The following steps are required when entering and releasing self-refresh mode.

1. Set the `SDSRF` bit to enter self-refresh mode.
2. Poll the `SDSRA` bit in the SDRAM status register (`SDSTAT`) to determine if the SDRAM has already entered self-refresh mode.
3. Set the `DSDCTL` bit to freeze `SDCLK` (optional).

Programming Models

4. Self refresh mode-no activities on all SDRAM signals (clock optional).
5. Clear the `DSDCTL` bit to re-enable `SDCLK` (optional).
6. SDRAM access releases controller from self-refresh mode.

Changing the VCO Clock During Runtime

In previous SHARC models, only a hardware reset initiated another SDRAM power-up sequence. This is no longer the case since the PLL allows programs to change the output clocks during runtime.

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

The reduction of the system clock violates the minimum specifications, while increasing the system clock violates the maximum t_{REF} specification. Therefore, careful software control is required to adapt these changes. Therefore, the release from self-refresh mode should be a dummy read operation since it happens with the old frequency settings.



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

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

1. Set the SDRAM to self-refresh mode by writing a 1 to the `SDSRF` bit of the `SDCTL` register.
2. Poll the `SDSRA` bit of `SDSTAT` register for self-refresh grant.

3. Execute the desired PLL programming sequence. (For more information, see “PLL Start-Up” on page 22-9.)
4. Wait 4096 CLKIN cycles ($\overline{\text{RESETOUT}}$ asserted) which indicates the PLL has settled to the new frequency.
5. Reprogram the SDRAM registers (SDRRC, SDCTL) with values appropriate to the new SDCLK frequency and assure that the SDPSS bit is set.
6. Bring the SDRAM out of self-refresh mode by performing a dummy read SDRAM access.
7. The SDC now issues the commands PREA, 8xREF and MRS to initialize the SDC and the SDRAM to the new frequency.

The SDRAM device is now ready to be accessed.

DDR2 Controller

The following sections are specific to DDR2 SDRAM memory on the ADSP-2146x processor.

Power-Up Sequence

The following steps are used to power-up the DDR2 device.

1. Ensure the minimum DDR2 clock frequency is stable and at least 125 MHz (according to datasheet).
2. Program the core to DDR2 clock ratio using the PMCTL register. For PLL changes wait at least 4096 core cycles, for output divider changes at least 15 core clock cycles for effect latency.
3. If a new frequency is desired, put the on-chip DLL into reset using DLL1-OCTL1 registers.
4. Wait at least 9 core cycles for the DLL to lock to a new frequency.

Programming Models

5. DLL in reset starts new locking event. Wait for the DLL to lock to the new frequency. Note DLL locking time depending on the CCLK to DDR2_CLK ratio is:
 - 1:2 – 3000 CCLK cycles
 - 1:3 – 7500 CCLK cycles
 - 1:4 – 10000 CCLK cycles
6. Assign the required external DDR2 banks in the EPCTL register.
7. Wait 8 core cycles for effect latency.
8. Program the refresh rate control register (DDR2RRC).
9. Program the timing parameters in the DDR2CTL1 register.
10. Program all MR and EMR3-1 settings in the DDR2CTL5-3 registers.
11. Enable DDR size (row, column, bank) in the DDR2CTL0 register.
12. Start the power-up sequence with the DDR2PSS bit. Wait for DLL external bank calibration.

The device now ready for any access.

Frequency Change in Precharge Power-Down Mode

This command allows programs to change the DDR2 clock during run time. Two different usage cases are relevant: changing the VCO or changing the core to DDR2 clock. Power-down requires careful software control since the DRAM is not refreshed. The clock frequency change must happen in a specific time window (typically $t_{RASmax} = 8 \times t_{REFI}$ or $9 \times t_{REFI}$). If you change the VCO frequency the $CLKIN \times 4096 < 9 \times t_{REFI}$ equation must be met. If it cannnot be met (most cases) the entire clock

frequency change requires self-refresh mode. The DDR2 DDR2 input core to DDR2 clock ratio change can be made under following condition:

1. The ODT must be turned off.
2. Put the DDR2 DDR2 in precharge power down mode (`DDR2CKE` pin goes low). A minimum of 2 DDR2 clock cycles must occur after the clock frequency change.
3. The core to DDR2 clock divider is allowed to change only within the minimum and maximum operating frequency specified for the particular speed grade. During input clock frequency change, ODT and `DDR2CKE` must be held at stable low levels.
4. Once the input clock frequency has changed, stable new clocks must be provided to DRAM before the precharge power down may be exited.
5. Reset the on-chip DLL and wait until it locks to new frequency.
6. Depending on new the clock frequency, an additional MRS or EMRS command may needed to appropriately set the writes, latencies and other parameters. During the DLL re-lock period, ODT must remain off.
7. Before writing to `DDR2CTL0` register, exit the precharge power-down mode (`DDR2CKE` pin goes high). This ensures the DLL reset via mode register happens during `DDR2CKE` high.

External Instruction Fetch

The section describes the software programming steps needed for the successful operation of external instruction fetch through the external port. Note only the additional steps for code execution are illustrated. For timing related settings refer to “[Functional Description](#)” on page 3-7.

Programming Models

AMI Configuration

For instruction fetch, the original (logical) address is multiplied by 3/2 and this address is translated depending on the bus width and `PKDIS` bit setting.

1. Assign external bank0 to AMI in the `EPCTL` register (default).
2. Wait at least 8 `CCLK` cycles (effect latency).
3. Enable the global `AMIVEN` bit and clear (=0) the `PKDIS` bit.

SDRAM Configuration

For instruction fetch, the original (logical) address is multiplied by 3/2 and this address is translated depending on the bus width setting (`X16DE` bit).

1. Assign external bank 0 to SDRAM in the `EPCTL` register (default).
2. Wait at least 8 `CCLK` cycles (effect latency).
3. Configure the `SDCTL` and `SDRRC` registers accordingly.

External Memory Access Restrictions

The following external memory restrictions should be noted when writing programs.

1. The `LW` mnemonic is not applicable to external memory.
2. Conditional accesses to external memory should not be based on any of the `FLAG` pin status.
3. There is one cycle latency between a multiplier status change and an arithmetic loop abort. This extra cycle is a machine cycle and not the instruction cycle. Therefore, if there is a pipeline stall (due to external memory access etc.) then the latency does not apply.

4. A one cycle stall is generated whenever an instruction that contains a conditional external memory access is in the decode stage, where the evaluation of the condition is dependent on the outcome of the previous instruction in address stage. It applies to all kinds of conditions except for conditions based on FLAG status. The following is an example:

```
f12 = f11+f10;  
if eq dm(ext) = r0;
```

5. The FLUSH CACHE instruction has an effect latency of one instruction when executing program instructions from internal memory, and two instructions when executing from external memory.
6. When a new external memory instruction fetch occurs on the processor due to a jump from internal to external memory, or after a cache hit while executing instructions from external memory, there is one stall cycle present in the fetch1 stage. This stall avoids resource conflicts at the cache interface.
7. Any sequence of external memory access (read or write) followed by an IOP access, causes the IOP access to fail. To workaround this restriction, separate the external memory access and IOP access by adding a NOP instruction or any other instruction which is not either an IOP read/write, or an external memory access. Example:

```
R12 = dm(Ext_mem);  
NOP; /* fixes restriction */  
R0 = dm(SPCTL2);
```

Programming Models

4 LINK PORTS—ADSP-2146X

The ADSP-2146x processors have two 8-bit wide link ports, which can connect to another processor or peripheral link ports. The link ports allow a variety of interconnection schemes to I/O peripheral devices as well as co-processing and multiprocessing schemes. The port specifications are shown in [Table 4-1](#).

Table 4-1. Link Port Specifications

Feature	Link Port1–0
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	Yes
Transmission Half Duplex	Yes
Transmission Full Duplex	No
Access Type	
Data Buffer	Yes

Features

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

Feature	Link Port 1–0
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	2
DMA Chaining	Yes
Boot Capable	Yes (Link Port 0)
Local Memory	No
Clock Operation	LCLK

Features

These bidirectional ports have eight data lines, an acknowledge line, and a clock line. The maximum frequency of operation of the link ports is 166 MHz. The link port clock to core clock ratio programming is applicable only if the link port is configured as transmitter. The receiver link port can operate at any asynchronous clock frequency up to 166 MHz (or peripheral clock frequency ($PCLK = CCLK/2$) which ever is lower) independent of the programmed ratio.

The link ports contain the features shown in the following list.

- Operate independently and simultaneously.
- Pack data into 32-bit words; this data can be directly read by the processor or DMA-transferred to or from on-chip memory.
- Have double-buffered transmit and receive data registers.

- Include programmable clock and acknowledge controls for link port transfers. Each link port has its own dedicated DMA channel.
- Provide high-speed, point-to-point data transfers to other processors, allowing differing types of interconnections between multiple DSPs.

Pin Descriptions

The pins associated with each link port are described in the ADSP-2146x data sheet.

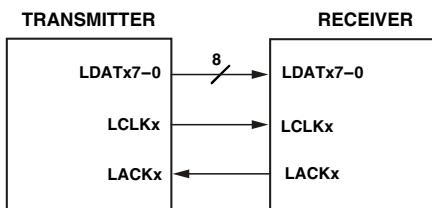


Figure 4-1. Link Port Pin Connections

Register Overview

Each link port has its own control and status register. These are described in the following sections and in “[Link Port Registers](#)” on page A-63. For information on the link port DMA registers, see “[Standard DMA Parameter Registers](#)” on page 2-4. For information on the link port buffer registers, see “[Data Buffers](#)” on page 2-10.

Control Registers (LCTLx). The control registers are used to enable the port, to set up DMA parameters, and to configure interrupts.

Clocking

Status Registers (LSTAT x). Programs can see several aspects of link port operation using the status registers. These include bus status, buffer status, receive and transmit status, and errors.

Clocking

The link port clock is derived from the clock out generator based on the linkport to core clock ratio. [For more information, see “Output Clock Generator” on page 22-5.](#)

The link port to core clock ratios (1:2, 1:2.5, 1:3, 1:4) can be programmed in the `PMCTL` register. This programming is applicable only for the transmitter. The receiver can operate at any asynchronous frequency up to the maximum frequency, independent of the ratio programmed.

Functional Description

Each link port, shown in [Figure 4-2](#), consists of eight data lines (`LDAT x 7-0`, $x = 0, 1$), a link port clock line (`LCLK x`), and a link port acknowledge line (`LACK x`). The `LCLK x` and `LACK x` pins of each link port allow handshaking for asynchronous data communication between DSPs. Other devices that follow the same protocol may also communicate with these link ports.

The link port operates in half-duplex mode, only receive or transmit operation can happen per linkport by using core or DMA. If full-duplex operation is required both linkport must be used.

In receive operation, the data are received by the external receive buffer packed into 32-bit format and shifted to the internal receive buffer. The core or DMA read the data from the internal buffer. In transmit operation, the data are written to the internal transmit buffer moved to the external transmit buffer to shift the data off-chip. The following sections provide details on theis interface.

Architecture

Figure 4-2 shows the architecture of the link ports.

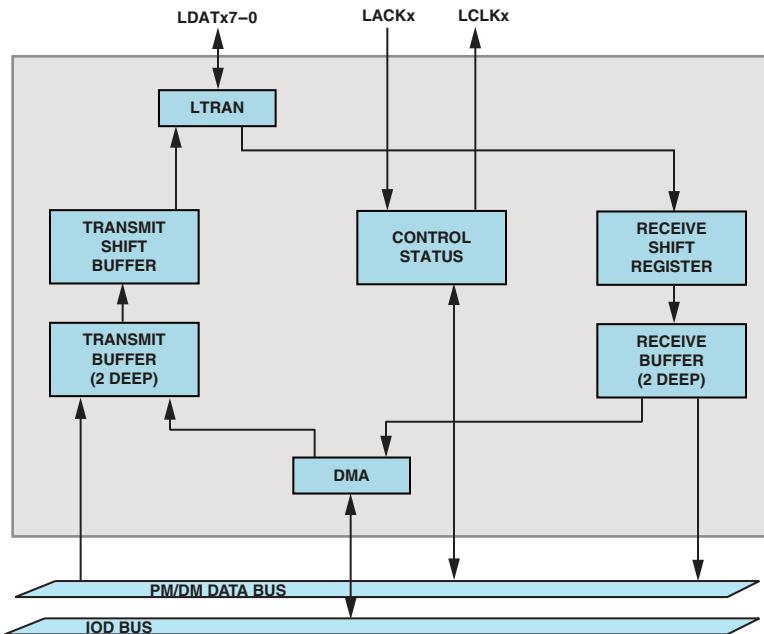


Figure 4-2. Link Port Block Diagram

Protocol

A link-port-transmitted word consists of 4 bytes (for a 32-bit word). The transmitter asserts the clock (**LCLKx**) high with each new byte of data. The falling edge of **LCLKx** is used by the receiver to latch the byte. The receiver asserts **LACKx** when it is ready to accept another word in the receive buffer, **RXLBx**. The transmitter samples **LACKx** driven by the receiver at the beginning of each word transmission (that is, after every 4 bytes with a positive level latch). If **LACKx** is deasserted at that time, the transmitter does not transmit the new word. The transmitter leaves **LCLKx** high and continues

Functional Description

to drive the first byte if `LACKx` is deasserted. When `LACKx` is eventually asserted again, the transmitter drives `LCLKx` low and begins transmission of the next word. If the transmit buffer is empty, `LCLKx` remains low until the buffer is refilled, regardless of the state of `LACKx`.

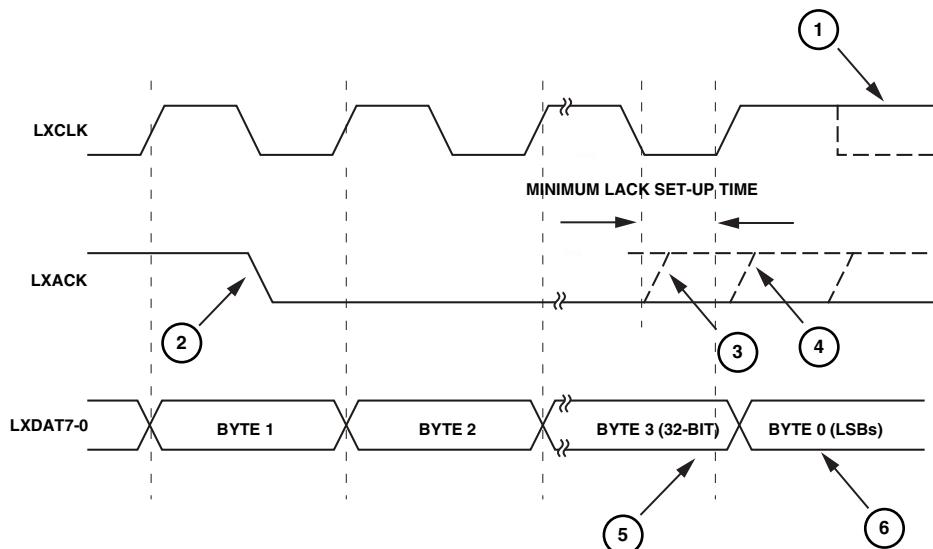


Figure 4-3. Link Port Handshake Timing

The following list describes the stages shown in [Figure 4-3](#).

1. `LCLK` stays high at byte 0 if `LACK` is sampled low on the previous `LCLK` rising edge. `LCLK` high indicates a stall.
2. `LxACK` may deassert after byte 0.
3. `LACK` reasserts as soon as the link buffer is not full.
4. Transmitter samples `LACK` to determine whether to transmit the next word.

5. Receiver accepts the remaining word even if LACK is deasserted. The transmitter does not send the following word.
6. Transmit data for next word is held until LACK is asserted.

The receive buffer may fill if a higher priority DMA, core I/O processor register access, or chain loading operation is occurring. The LACK_x signal may deassert when it anticipates the buffer may fill. The LACK_x signal is reasserted by the receiver as soon as the internal DMA grant signal has occurred, freeing a buffer location or the core reads the receive buffer RXLB_x thereby freeing a buffer location. The LACK_x signal inhibits transmission of next word and not of the current byte.



Data is latched in the receive buffer on the falling edge of LCLK_x. The receive operation is purely asynchronous and can occur at any frequency up to 166 MHz or peripheral clock frequency (whichever is less).

When a link port is not enabled, LDAT7-0, LCLK_x and LACK_x are three-stated. When a link port is enabled to transmit, the data pins are driven with whatever data is in the output buffer, LCLK_x is driven high and LACK_x is three-stated. When a port is enabled to receive, the data pins and LCLK_x are three-stated and LACK_x is driven high.

Intercommunication

The transmitter and the receiver may be enabled at different times. The LACK_x and LCLK_x signals should be held low with the external pull-down resistors. If the transmitter is enabled before the receiver, the LACK_x signal (of the receiver) is held low and transmission is held off. If the receiver is enabled before the transmitter, the LCLK_x signal (of the transmitter) is held low by the pull-down and the receiver is held off.

Functional Description

i Unlike older SHARC processors that have link ports, the ADSP-2146x processors do not have an internal pull-down resistor. Because of this there is no PDRDE bit available to disable the internal pull-down and an external pull-down (20K Ohms) is required on the LACKx and LCLKx signals.

[Figure 4-4](#), [Figure 4-5](#) and [Figure 4-6](#) show various timings for the link port.

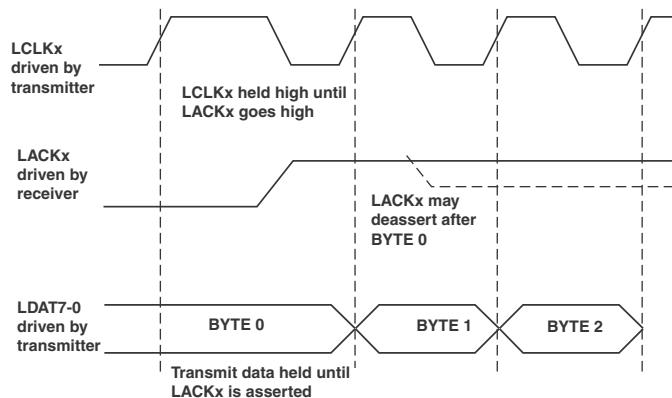


Figure 4-4. Enable Transmitter and Receiver at Different Times

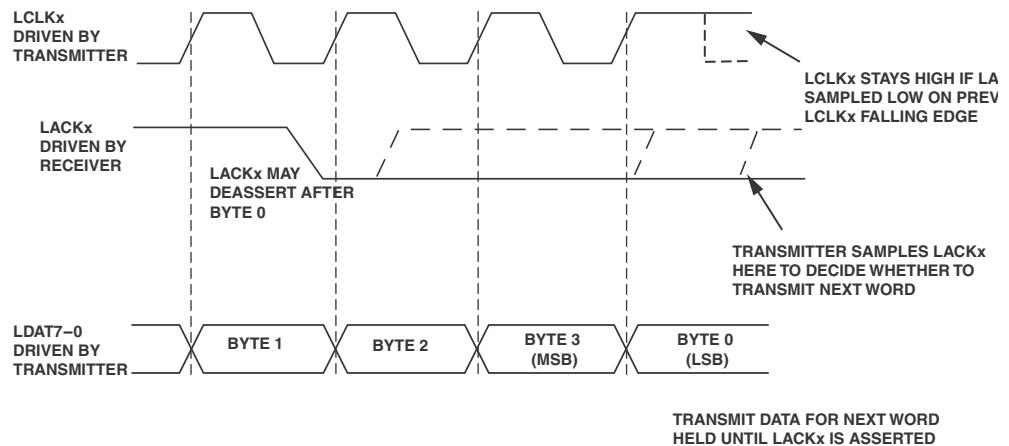


Figure 4-5. Relationship Between ACK and LCLK

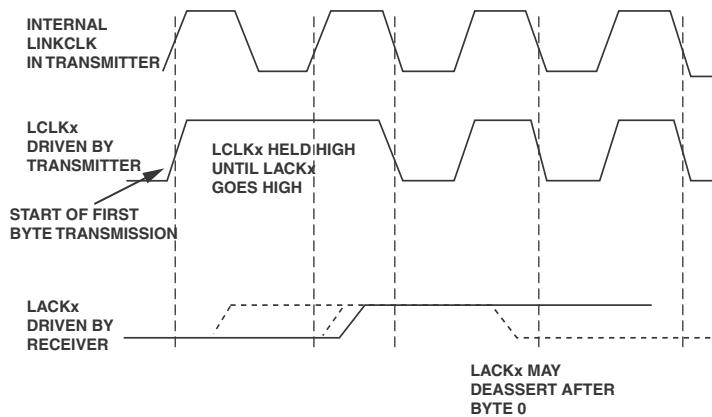


Figure 4-6. Relationship Between Internal LCLK and LCLK at Pads

Functional Description

Self-Synchronization

The link ports are designed to allow long distance connections to be made between the driver and the receiver. This is possible because the links are self-synchronizing—the clock and data are transmitted together. Only relative delay, not absolute delay between clock and data is relevant.

In addition, the `LACKx` signal inhibits transmission of the next word, not of the current nibble or byte. For example, the current word is always allowed to complete transmission. This allows delays of 3 cycles for the `LACKx` signal to reach the transmitter.

Multi-Master Conflicts

Multi-master conflicts can be resolved using token passing. In token passing, the token is a software flag that passes between processors. At reset, the token is set to reside in the link port of one device, making it the master and the transmitter. When a receiver (slave) wants to become the master, it may assert its `LACKx` line to get the master's attention. The master knows, through software protocol, whether it is supposed to respond with actual data or whether it is being asked for the token. If the master wishes to give up the token, it may send back a user-defined token release word and thereafter clear its token flag. Simultaneously, the slave will set its token and can thereafter transmit.

This example shown in [Figure 4-7](#) is a typical case where the link port is used as fast IO link. A FPGA bridge is required to communicate between two different protocols. If using both link ports, full duplex operation is possible without core intervention.



Figure 4-7. Fast I/O Link

Example Token Passing

When two ADSP-2146x processors communicate using a link port only one can be the transmitter or receiver. Token passing is a protocol that assists the DSPs alternate control. [Figure 4-8](#) shows a flow chart of the token passing process.

In token passing, the token is a software flag that passes between the processors. At reset, the token (flag) is set to reside in the link port of one device, making it the master and the transmitter. When a receiver link port (slave) wants to become the master, it may assert its LACK_x line (request data) to get the master's attention. The master knows, through software protocol, whether it is supposed to respond with actual data or whether it is being asked for the token.

The token release word can be any user-defined value. Since both the transmitter and receiver are expecting a code word, this does not need to be exclusive of normal data transmission.

If the master wishes to give up the token, it may send back a user-defined token release word and thereafter clear its token flag. Simultaneously, the slave examines the data sent back and if it is the token release word, the slave will set its token, and can thereafter transmit. If the received data is not the token release word, then the slave must assume the master was beginning a new transmission.

Through software protocol, the master can also request data by sending the token release word (TRW) without the LACK_x (data request) going low first.

Functional Description

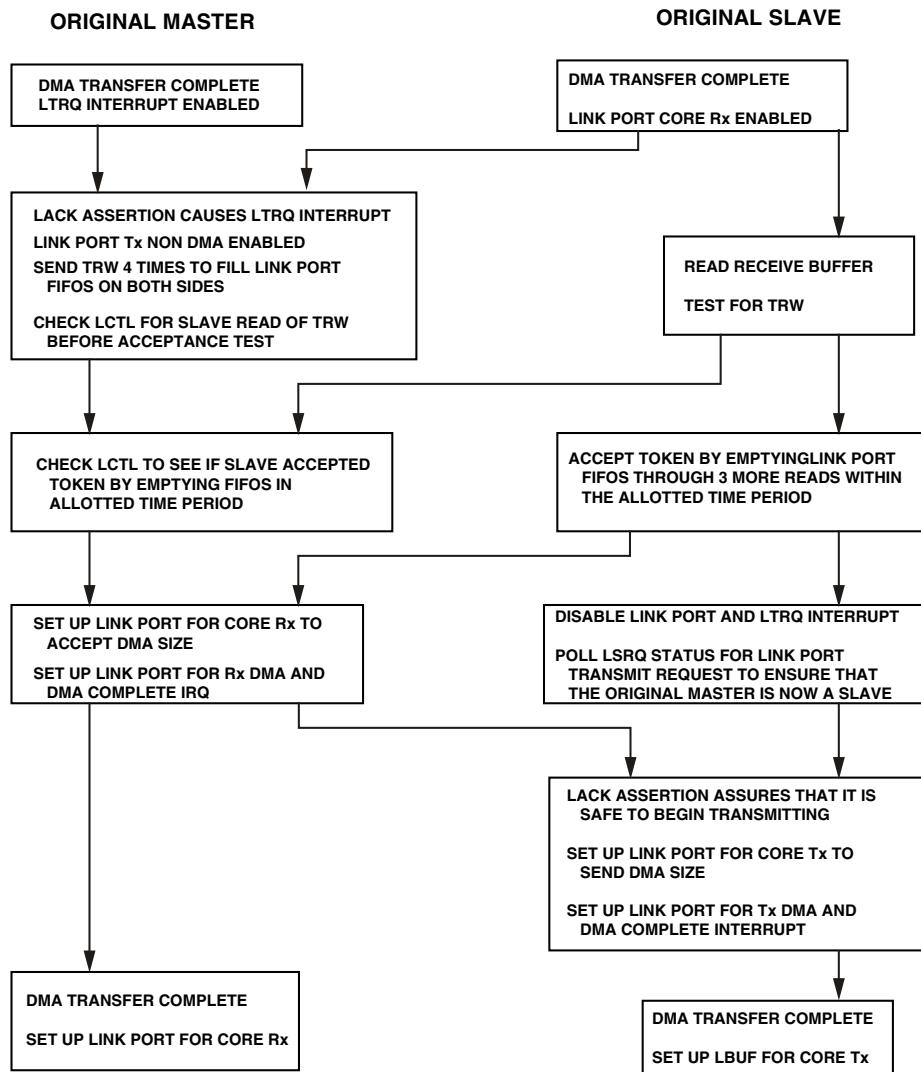


Figure 4-8. Token Passing Flow Chart

The following is a list of the areas of concern when a program implements a software protocol scheme for token passing:

- The program must make sure that both link ports are not enabled to transmit at the same time. In the event that this occurs, data may be transmitted and lost due to the fact that neither link port is driving `LACKx`. In the example, the `TLRQ` status bit is polled to ensure that the master becomes the slave before the slave becomes the master, avoiding the two transmitter conflict.
- The program must make sure that the link interrupt selection matches the application. If a status detection scheme using the status bits is to be used, it is important to note the following: If a link port that is configured to receive is disabled while `LACKx` is asserted, there is an RC delay before the external pulldown resistor on `LACKx` (if enabled) can pull the value below logic threshold. If the `LTRQ` status bit is unmasked (in this instance), then an LSR is latched and the `LSRQ` interrupt may be serviced, even though unintended, if enabled.
- The program must make sure that synchronization is not disrupted by unrelated influences at critical sections where timing control loops are used to synchronize parallel code execution. Disabling of nested interrupts is one technique to control this.

Data Transfer

The link ports are able to transfer data using DMA and core.

Link Buffers

The transmit buffer registers (`TXLBx`) and receive buffer registers (`RXLBx`) buffer the data flow through the link port. The transmit and receive

Data Transfer

buffers consist of a 2 deep buffer and a shift register. The registers read from or write to internal memory under DMA or processor core control.

Transmit Buffer

In the transmit path, the buffer is used to accept core data or DMA data from internal memory. Data is transferred to the shift register to send unpacked bytes to the ports. The least significant byte is transmitted first. As each word is unpacked and transmitted, the next location in the FIFO becomes available and a new DMA request is made if DMA is enabled. If the shift register becomes empty, the `LCLKx` signal is deasserted.

Receive Buffer

In the receive buffer, data is transferred to the core or DMA from the buffer whereas the shift register performs the packing, least significant byte first (the least significant byte is placed in bits 7–0). The `LACKx` signal is deasserted by the receiver as soon as it receives the first byte from transmitter if the buffer already has a word (the receive buffer `RXLBx` is already half full). The packing is done as shown in [Figure 4-9](#).

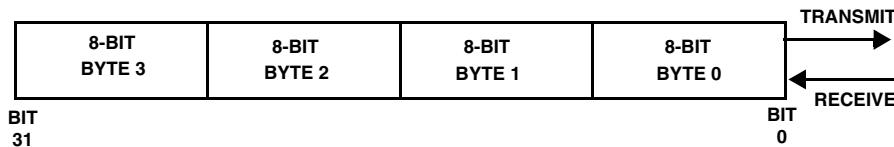


Figure 4-9. Transmit and Receive Buffers



For the ADSP-2146x processor, the least significant byte is transmitted first. This is different to legacy processors (ADSP-2116x) where the most significant byte is transmitted first.

Buffer Status

The entire receive and transmit path form a 3-stage FIFO. Two writes/reads can occur to the transmit/receive buffer by the core or DMA before it signals a full/empty condition. Full/empty status for the link buffer is shown by the FFST bits in the LSTAT_x register. If the link port is configured as a transmitter, then the FFST bits in the LSTAT_x register reflect the status of the TXLB_x register. If the link port is configured as a receiver, then the FFST bits in LSTAT_x register reflect the status of RXLB_x.

Core Transfers

In applications where the latency of link port DMA transfers to and from internal memory is too long, or where a process is continuous and has no block boundaries, the processor core may read or write link buffers directly using the full or empty status bit of the link buffer to automatically pace the operation. The full or empty status of a particular link buffer can be determined by reading the LSTAT_x bits in the LCTL register.



DMA should be disabled if reading or writing to the link port buffers.

If a read is attempted from an empty receive buffer, the core stalls (hangs) until the link port completes reception of a word. If a write is attempted to a full transmit buffer, the core stalls until the external device accepts the complete word. Up to four words (2 in the receiver and 2 in the transmitter) may be sent without a stall before the receiver core or DMA must read a link buffer register.

To support debugging buffer transfers, the processor has a buffer hang disable (LP_BHD) bit. When set (=1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition.

DMA Transfers

Each link port supports a DMA channel.

 Note that the link ports do not support internal to internal memory transfers like previous SHARCs (no link assignment register). If internal to internal memory transfers are required, refer to “[External Port DMA](#)” on page 3-65.

In standard DMA operations, the software needs to set up the DMA parameter registers before the link port control register is configured. After setting the DMA enable bit the transfer starts until the word count reaches zero, the DMA has finished.

Interrupts

The following sections and [Table 4-2](#) provide details on using link port interrupts.

Table 4-2. Link Port Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
Link port Core DMA	–DMA RX/TX done –core RX buffer full –core TX buffer empty –link service request –invalid transmit attempt	–internal transfer completion –access completion	RTI instruction	Need to route LPxI (PICRx) to any PxxI

Interrupt Sources

Five types of interrupts are dedicated to link ports.

1. A DMA channel interrupt is generated when a DMA block transfer through the link port with DMA enabled completes.
2. A DMA channel interrupt is generated when DMA for the link buffer channel is disabled and the buffer is not full or the buffer is not empty.
3. A link service request interrupt is generated when an external source accesses the link port when the link port is disabled. ($LxTRQ$ and $LxRRQ$). For example, if the enabled receiver wants to initiate a data transfer with the disabled transmitter, it can make $LACKx$ high. When $LACKx$ of the disabled link port goes high, then a link service request interrupt is generated. Now the receiver can initiate the transfer.
4. A link port invalid transmit LPIT is generated if the transmitter is driving $LCLKx$ high because the receiver has not asserted $LACKx$ and $LCLKx$ goes low due to a processor reset (or some other reason, even though the receiver has not yet asserted $LACKx$). In this case, the receiving link port generates an interrupt.
5. The transmitter generates an external transfer done interrupt once the external transfer is completed. When DMA is not enabled, this interrupt is generated when the transmitter FIFO is empty and the last byte has been transmitted. If using DMA, the transmitter checks if the DMA is complete.

Interrupt Service

All interrupts are latched and stored in the corresponding status register. Whenever any of these four interrupts occur, the corresponding interrupt LP0I or LP1I occurs if it is unmasked. In the ISR, the user should read the

Interrupts

corresponding status register (`LSTATx`). Reading the status register clears the interrupt bits. Read of the status register when an interrupt occurs causes the core to hang till the interrupt bits are set in the status register. Otherwise simultaneous read of the status register and updating of the status register will result in loss of information. This hang cannot be overridden with the `BHD` bit `LPCTLx` register.

Access Completion

An external transfer complete interrupt is generated by the transmitter once the external transfer is completed by setting bit 12 (`EXTTXFR_DONE_MSK`) in the `LCTL` register. When DMA is not enabled, this interrupt is generated when the transmitter FIFO is empty and the last byte has been transmitted. Also, when DMA is enabled, the DMA engine checks if DMA has been completed. (If `CLBx` is zero, and chaining is enabled, the DMA engine also checks if `CPLBx` also is zero.)

Internal Transfer Completion

This interrupt performs like previous SHARC processors where an internal transfer complete interrupt is generated by the transmitter once the word count is zero by setting bit 10 (`DMACH_IRPT_MSK`) in the `LCTL` register. When DMA is not enabled, this interrupt is generated when the word count is zero. Also, when DMA is enabled, the DMA engine checks if DMA has been completed. (If `CLBx` is zero, and chaining is enabled, the DMA engine also checks if `CPLBx` also is zero.)



For correct operation, programs should not perform simultaneous reads and updates of the status register as this results in loss of information. When an interrupt occurs, reads of the status register cause the core to hang until the interrupt bits are set in the status register. This hang cannot be overridden with the `LP_BHD` bit in the `LPCTLx` register.

DMA Access

A link port interrupt is generated when the DMA operation is done—when the block transfer has completed and the DMA count register is zero.

One way programs can use this interrupt is to send additional control information at the end of a block transfer. Because the receive DMA buffer is empty when the DMA block has completed, the external bus master can send up to two additional words to the slave processor's buffer, which has space for the two words. When the slaves's DMA completes, there is an interrupt. In the associated interrupt service routine, the buffer can be read in order to use these control words to determine the next course of action.

Chained DMA

In chained DMA operations, the processor automatically sets up another DMA transfer when the current DMA operation completes. The chain pointer register points to the next set of DMA parameters stored in the processor's internal memory. Each new set of parameters is stored in a four word, user-initialized buffer in internal memory known as a transfer control block (TCB). In TCB chain loading, the processor's IOP automatically reads the TCB from internal memory and then loads the values into the channel parameter registers to set up the next DMA sequence. The chain pointer can be loaded at any time during the DMA sequence. Writing all zeroes to the address field of the chain pointer also disables chaining.

Core Access

If DMA is disabled for a link port buffer, then the buffer may be written or read by the processor core as a memory-mapped I/O processor register.

Interrupts

If the DMA is disabled but the associated link buffer is enabled, then a maskable interrupt is generated whenever a receive buffer is not empty or when a transmit buffer is not full. This interrupt is the same interrupt vector associated with the completion of the DMA block transfers.

The interrupt latch bit may be unmasked by the corresponding mask bit in the same register. When initially enabling the mask bit, the corresponding latch bit should be cleared first to clear out any request that may have been inadvertently latched.

The interrupt service routine should test the buffer status or write to check when the buffer is empty or full, in order to determine when it should return from interrupt. This reduces the number of interrupts the ISR must service.

For core interrupts programs need to perform two writes to the transmit buffer to fill it. For the receiver, programs need to perform two reads from the receive buffer to empty it. Then the next interrupt occurs when the buffer becomes not full or not empty again. Two reads and or writes are performed because the link port FIFO depth is two.

Service Request Interrupts

Link port service requests let a disabled (unassigned or assigned with buffer disabled) link port cause an interrupt when an external access is attempted. The transmit (`LTRQ`) and receive (`LRRQ`) request status bits of the `LCTLx` register indicate when another processor is attempting to send or receive data through a particular link port. Two processors can communicate without prior knowledge of the transfer direction, link port number, or exactly when the transfer is to occur. The `LCTLx` register is described in `regs` appendix.

When `LxACK` or `LxCLK` is asserted externally, a link service request (LSR) is generated in a disabled (unassigned or assigned with buffer disabled) link port. Each LSR is gated by mask bits before being latched in the `LSTATx` register. The two possible receive LSRs and the two possible transmit

LSRs are gated by mask bits and then ORed together to generate the link service request interrupt. The LSRQ interrupt requests may be masked by the receive (LRRQ_MSK) or transmit (LTRQ_MSK) bits of the LCTLX register.

When the mask bit is set, the interrupt is allowed to pass into the interrupt priority encoder. The maximum latency between asserting the LCLK or LACK signals and latching an interrupt is 2 to 3 PCLK cycles.

The interrupt routine must read the LSTATX register to determine which link port to service and whether it is a transmit or receive request. The link service request status of the port is set whenever the port is not enable and one of LxACK or LxCLK is asserted high.

If link service requests are in use, they should be masked out when the assigned link buffers are being enabled, disabled, or when the link port is being unassigned in LCTLX register. Otherwise, spurious service requests may be generated.



To avoid the possibility of spurious interrupts, programs can mask the LSRQ interrupts and poll the appropriate request status bits until it is cleared and then unmask the interrupt.

Debug Features

The following sections provide information on features that help in debugging link port software.

Shadow Register

The shadow status register can be read without clearing the interrupt bits. For more information, see “Link Port Registers” on page A-63.

Effect Latency

Buffer Hang Disable (BHD)

A buffer hang disable (BHD) bit has been provided in the control register (LPCTLx). Setting this bit to 1 prevents the core from hanging when a read from an empty receive buffer or a write to a full transmit buffer is attempted. If the BHD bit is set and a read is performed from an empty receive buffer, then the previous data is returned. Writing to a full transmit buffer with the BHD bit set overwrites the existing data.

Effect Latency

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

Write Effect Latency

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

Link Port Effect Latency

After the link port registers are configured the effect latency is 2 PCLK cycles.

Programming Model

The following sections provide information on programming receive and transmit DMA and changing the link port clock.

Changing the Link Port Clock

The following programming sequence may be used to change the core-to-link port clock ratio only. Note that this procedure changes only the PLL output divider. Therefore programs do not need to wait 4096 `CLKIN` cycles (required only if the PLL multiplier or the `INDIV` bit is modified).

1. Disable the link ports. Note that the peripherals cannot be enabled when changing clock ratio.
2. Select the PLL divider by setting the `PLLDx` bits (bits 6–7 in the `PMCTL` register).
3. Select the link port clock divider (`CCLK` to `LPCLK` ratio) by setting the `LPCKRx` bits (bits 21 and 22 in the `PMCTL` register).
4. Enable the new divisors by setting the `DIVEN` bit (bit 9 in the `PMCTL` register).
5. Wait 15 `CCLK` cycles. During this time, programs must not execute any valid instructions. The `LPCLK` change does not happen on-the-fly. This means that when a clock ratio change is registered, the current clock cycle may get truncated before the change and the new clock cycle ratio start.
6. Enable link ports.

For more information on link port clocking and programming the PLL, see “[Phase-Locked Loop \(PLL\)](#)” on page [22-2](#).

Receive DMA

The following is the sequence that occurs when an external device transfers a block of data into the processor's internal memory using a link port.

 Note that the link ports do not support internal to internal memory transfers like previous SHARCs. If internal to internal memory transfers are required, refer to “[External Port DMA](#)” on page 3-65.

1. The processor writes the DMA channel's parameter registers (index register `IILBx`, modify register `IMLBx` and count register `CLBx`) and initializes the link port for receive (`LTRAN = 0`).
2. The processor enables the link port by setting the `LEN` bit. DMA is enabled by setting the `LDEN` bit in the `LCTLx` register.
3. The external device begins writing data to the `RXLBx` buffer through the link port.
4. The `RXLBx` buffer detects that data is present and sends a internal DMA request.
5. After the request is granted, the DMA transfer is performed thereby emptying the `RXLBx` buffer FIFO.

Transmit DMA

The following is the sequence that occurs when the processor transfers a block of data from its internal memory to an external device using link port.

1. The processor writes the DMA channel's parameter registers (index register `IILBx`, modify register `IMLBx` and count register `CLBx`) and initializes the link port for transmit (`LTRAN = 1`).

2. The processor enables the link port by setting the LEN bit and enables the link port DMA by setting the LDEN bit in the LCTL_x register. Because this is a transmit, setting LDEN automatically asserts an internal DMA request.
3. After the request is granted the internal DMA transfer is performed filling the TXLB_x buffer's FIFO.
4. The external device begins reading data from the TXLB_x buffer through the link port.
5. The TXLB_x buffer detects that there is room in the buffer (partially empty) and asserts another DMA request continuing the process.

Programming Model

5 MEMORY-TO-MEMORY PORT DMA

[Table 5-1](#) shows the memory-to-memory DMA port specifications.

Table 5-1. MTM Port Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	No
Transmission Simplex	Yes
Transmission Half Duplex	No
Transmission Full Duplex	No
Access Type	
Data Buffer	Yes
Core Data Access	No
DMA Data Access	Yes
DMA Channels	2
DMA Chaining	No

Features

Table 5-1. MTM Port Specifications (Cont'd)

Feature	Availability
Boot Capable	No
Local Memory	No
Clock Operation	f_{PCLK}

Features

The memory-to-memory port incorporates:

- 2 DMA channels (read and write)
- Internal to internal transfers
- Data engine for DTCP applications (only for special part numbers)

Note that the SHARC supports another internal to internal DMA module (external port) which does support multiple DMA modes.

Register Overview

MTM control register (MTMCTL). Enables the read and write DMA channels across the internal memory and returns status about the read or write DMA channel.

Clocking

The fundamental timing clock of the MTM is peripheral clock ($PCLK$).

Functional Description

The MTM module owns two DMA channels one for read and one for write including a data buffer which stores up to 2x32-bit data. After the DMA is configured, the read DMA channel fills the buffer with 64-bit data. After this transfer, the write DMA channel becomes active and empties the buffer according to its destination. This procedure is repeated until the DMA count is zero.

The memory-to-memory DMA controller is capable of transferring 64-bit bursts of data between internal memories.



The MTM controller supports data in normal word address space only (32-bit). External to external DMA transfers are not supported.

Data Transfer Types

The memory-to-memory DMA controller is capable of transferring 64-bit bursts of data between internal memories.

Data Buffer

The `MTMFLUSH` bit in the `MTMCTL` register can be set to flush the FIFO and reset the read/write pointers. Setting and resetting the `MTMDEN` bit only starts and stops the DMA transfer, so it is always better to flush the FIFO along with `MTMDEN` reset.

Note that the `MTMFLUSH` bit should not be set along with the `MTMDEN` bit set. Otherwise the FIFO is continuously flushed leading to DMA data corruption.

Interrupts

DMA Transfer

Two DMA channels are used for memory-to-memory DMA transfers. The write DMA channel has higher priority over the read channel. The transfer is started by a write DMA to fill up the MTM buffer with a 2 x 32-bit word. Next, the buffer is read back over the same IOD bus to the new destination. With a two position deep buffer and alternate write and read access over the same bus, throughput is limited. The memory-to-memory DMA control register (`MTMCTL`) allows programs to transfer blocks of 64-bit data from one internal memory location to another. This register also allows verification of current DMA status during writes and reads.

Interrupts

There are two DMA channels; one write channel and one read channel. When the transmission of a complete data block is performed, each channel will generate an interrupt to signal that the entire block of data has been processed. Note that the write and read interrupts (`P15I`, if the `MTMI` bit in the `IMASK` register is enabled) are very close to each other, so only one interrupt is triggered.

[Table 5-2](#) provides an overview of MTM interrupts.

Table 5-2. MTM Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
MTM (2 channels)	– WR DMA done – RD DMA done	Internal transfer completion	RTI instruction	<code>P15I</code>

MTM Throughput

Data throughput for internal to internal transfers is 12 PCLK cycles for 64-bit data.

Effect Latency

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

Write Effect Latency

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

MTM Effect Latency

After the MTM register is configured the effect latency is 1.5 PCLK cycles minimum and 2 PCLK cycles maximum.

Programming Model

This data transfer can be set up using the following procedure.

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

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

Programming Model

6 FFT/FIR/IIR HARDWARE MODULES

Finite Impulse Response (FIR) filters are frequently used in DSP applications. With its high performance floating-point processing capabilities the SHARC processors are uniquely designed for FIR filtering. The SIMD SHARC core has two MAC units which provide 800 MIPS of processing speed when the processor is running at 400 MHz. However, for high performance applications, with their ever increasing complexity (such as room equalization or surround sound), even more processing power is needed.

To meet this need, the ADSP-214xx SHARC processors off load some of the most frequently used and intensive processing into hardware accelerators. An accelerator dedicated for filter processing can reduce the instruction processing load on the core, freeing it up for other tasks.

The FIR/IIR/FFT accelerator units are capable of performing the filters and FFT without core intervention. This gives software developers enormous freedom to use core processing cycles to implement complex algorithms, effectively adding more instructions per second to the processor.



The accelerator modules (FFT/FIR/IIR) each have local memory which is not accessible by the core during regular operation mode.

The interface specifications are shown in [Table 6-1](#).

Table 6-1. Accelerator Specifications

Feature	FFT/FIR/IIR
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	N/A
Slave Capable	N/A
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A
Access Type	
Data Buffer	Yes
Core Data Access	No
DMA Data Access	Yes
DMA Channels	2
DMA Chaining	Yes
Boot Capable	N/A
Local Memory	Yes (RAM)
Clock Operation	f_{PCLK}

FFT Accelerator

The FFT accelerator (shown in Figure 6-1) implements radix-2 complex floating-point FFT. The accelerator's data and twiddle coefficient interface is designed to connect to the processor's DMA engine (acting like a peripheral) and implements a synchronous pipeline read/write protocol with a pipeline depth of 1.

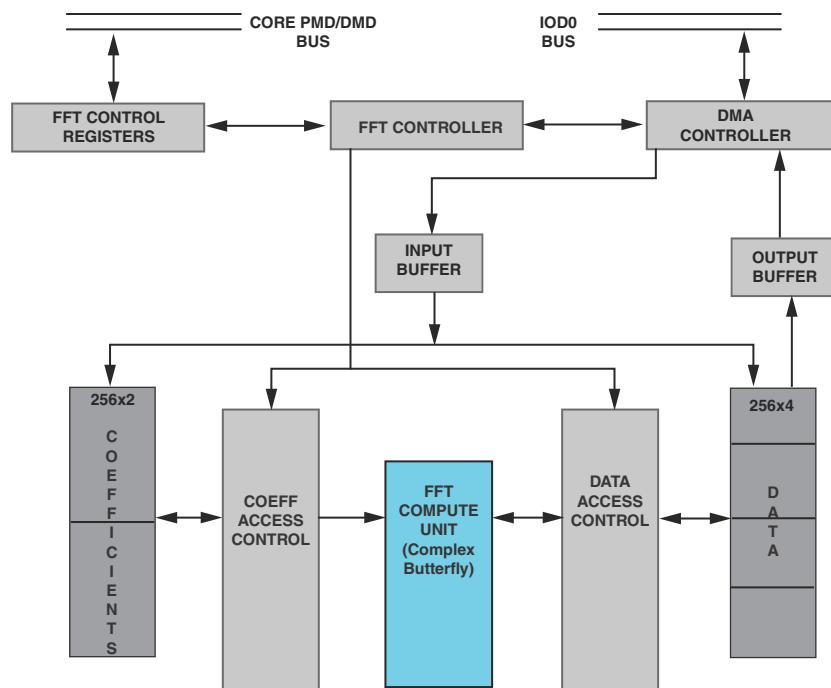


Figure 6-1. FFT Block Diagram

Features

The following list describes the features available through the FFT accelerator.

- Supports FFT sizes from $16 - 8k^2$ points all handled by DMA with no core intervention.
- Computes a radix 2 decimation in time algorithm with automated bit reversal.
- Contains a 1024 32-bit word data memory unit.
- Contains a 512 32-bit word twiddle coefficients memory unit.
- Contains a compute block unit with four floating-point multipliers and six floating-point adders.
- Has a control unit with configuration registers, responsible for all memory addresses and strobe generation.
- Contains a 8×32 deep input/output FIFO unit.

Register Descriptions

The accelerator has two control and two status registers that are used to program and check operation of the module. The module also contains DMA registers which are described in “[I/O Processor](#)” in Chapter 2, I/O Processor.

Power Management Control Register (PMCTL1). Used for FFT accelerator selection. Controls the clock power down to the module if not required.

Global Control Register (FFTCTL1). Used to enable, start, and reset the FFT module. It is also used to enable DMA and debug operation.

Control Register (FFTCTL2). Used to configure individual FFT parameters (such as length) and how the module process the FFT, such as data packing.

MAC Status Register (FTTMACSTAT). Reports errors and status on the multiply/accumulator.

DMA Status, Shadow DMA Status Registers (FTTDMASTAT, FFTSHDMASTAT). Provide information on DMA operations such as DMA progress and chain pointer loading.

Clocking

The FFT accelerator runs at the maximum speed of the peripheral clock (f_{PCLK}).

Functional Description

The FFT accelerator is comprised of a compute block, data memory and coefficient memory. The design allows programs to offload an FFT calculation by initializing few TCBs and control registers. In this way, the FFT accelerator can perform the FFT calculation in the background while the core is busy doing some other useful task. It can interrupt the core once the processing is complete. The following sections provide functional details of the FFT accelerator.

Compute Block

The compute block contains one complex butterfly stage (based on four IEEE floating-point multipliers and six IEEE floating-point adders) whose operation is pipelined and simultaneous.

Data Memory

The accelerator has a 1024 location deep, 32-bit wide data memory, organized into four independent blocks. Blocks are grouped in sets of two that are used to fetch or store real and imaginary parts of data simultaneously. Fetches and stores are accomplished by ping-ponging the read and write buffers.

Coefficient Memory

The accelerator has a 512 location deep, 32-bit wide twiddle memory, organized into two independent blocks (256x2). It allows fetching real and imaginary twiddles simultaneously.

Accelerator States

The FFT accelerator has five different states:

1. Reset
2. Idle
3. Reading
4. Processing
5. Writing

These states are described in detail in the following sections.

Reset State

Reset mode is activated either by setting the `FFT_RST` bit in the `FFTCTL1` register or by applying logic low to the `RESET` input pin.

If reset is activated by setting the `FFT_RST` bit, this bit must be cleared to bring the accelerator out of reset.

Resetting via a logic low to the `RESET` pin resets all registers, thereby clearing the `FFT_RST` bit. Once the processor is brought out of reset by applying a logic high to the `RESET` pin, the FFT module goes into the idle state in the next clock cycle.

Idle State

This mode is used to program the accelerator's control registers. Setting the `FFT_EN` and `FFT_START` bits in the `FFTCTL1` register moves the state from idle to reading.

Read State

In this state the module reads data and coefficients, but counts the number of read data only. This is because for successive FFT calculations the coefficient need not be read again—only the next set of data has to be read. When a specified number of data words are read, the state automatically moves to processing.

Processing State

In this mode the module computes FFT ping-pong stages in memory. Once this is done, the state automatically moves to the write state.

Write State

In this mode all the computed data is written out to internal memory. The state then automatically changes to either idle or read, depending on the way the block is configured using the repeat function (`FFT_RPT` bit in the `FFTCTL2` register). If the `FFT_RPT` bit is set, the block moves to the read state, if cleared, the block moves to the idle state. The `FFT_RPT` bit is useful when programs need to continuously perform an FFT on input data without core intervention.

Internal Memory Storage

This section describes the required software buffers in internal memory and the required storage model for data and coefficients using the FFT accelerator.

Small FFT N<=256

To run a small FFT three buffers are required:

- Input Buffer [$2 \times N$] packed or unpacked data
- Output Buffer [$2 \times N$] packed or unpacked data
- Coefficient Buffer [$2 \times N$]

Unpacked Data

If unpacked data is selected this is the required input format or output format. Programs can optionally select the input or output data streams to be unpacked. In this mode, the first samples are all real followed by the imaginary samples.

RE[0], RE[1], ... RE[N-1], IM[0], IM[2], ... IM[N-1]

This can be independent for the input or output data streams.

Packed Data

The default format for packed data is as follows.

RE[0], IM[0], RE[1], IM[1], ..., RE[N-1], IM[N-1]

Twiddles

The default format for coefficient buffer (twiddles) is as follows.

Re(CF[0]), Im(CF[0]), -Im(CF[0]), Re(CF[0]), Re(CF[1]),
Im(CF[1]), -Im(CF[1]), Re(CF[1]), Re(CF[N/2-1]),
Im(CF[N/2-1]), -Im(CF[N/2-1]), Re(CF[N/2-1]) (4xN/2 = 2N words)]

Large FFT N>=256

To run a large FFT 7 buffers are required:

- Input Buffer [2 × N] (packed data)
- Special Buffer [2 × N] (intermediate buffer used in step 1 for vertical FFT and in step 2 for special product = Product of vertical buffer with special twiddles)
- Output Buffer [2 × N] (packed data)
- Vertical Coeff Buffer [2 × V]
- Horizontal Coeff Buffer [2 × H]
- Special Coeff Buffer [4 × N]

Twiddles

For N>256, the FFT accelerator follows the ‘Divide and Conquer’ approach. Therefore, three types of coefficient buffers are required:]

Coefficient buffer for V point FFT

```
Re(CF[0]), Im(CF[0]), -Im(CF[0]), Re(CF[0]),
Re(CF[1]), Im(CF[1]), -Im(CF[1]), Re(CF[1]),
.....
Re(CF[V/2-1]), Im(CF[V/2-1]), -Im(CF[V/2-1]), Re(CF[V/2-1])
(4xV/2 = 2V words)
```

Coefficient buffer for H point FFT

```
Re(CF[0]), Im(CF[0]), -Im(CF[0]), Re(CF[0]),
Re(CF[1]), Im(CF[1]), -Im(CF[1]), Re(CF[1]),
.....
Re(CF[H/2-1]), Im(CF[H/2-1]), -Im(CF[H/2-1]), Re(CF[H/2-1])
(4xH/2 = 2H words)
```

Special coefficient buffer

$\text{Re}(\text{SP_CF}[0]), \text{Im}(\text{SP_CF}[0]), -\text{Im}(\text{SP_CF}[0]), \text{Re}(\text{SP_CF}[0]),$
 $\text{Re}(\text{SP_CF}[1]), \text{Im}(\text{SP_CF}[1]), -\text{Im}(\text{SP_CF}[1]), \text{Re}(\text{SP_CF}[1]),$
.....
 $\text{Re}(\text{SP_CF}[N-1]), \text{Im}(\text{SP_CF}[N-1]), -\text{Im}(\text{SP_CF}[N-1]), \text{Re}(\text{SP_CF}[N-1])$
(4N words).

Where,

$\text{Re}(\text{CF}[x])$ = Real part of the complex coefficient $\text{CF}[x]$,

$\text{Im}(\text{CF}[x])$ = Imaginary part of the complex coefficient $\text{CF}[x]$,

$\text{Re}(\text{SP_CF}[x])$ = Real part of special complex coefficient $\text{SP_CF}[x]$,

$\text{Im}(\text{SP_CF}[x])$ = Imaginary part of special complex coefficient $\text{SP_CF}[x]$,

$n = v \times H + h, n = 0, 1$

...

$N - 1$

$$\text{CF}[x] = W_N^x$$

$$SP_{CF[n]} = \sum_0^{V-1} \sum_0^{H-1} W_N^{v \times h}$$

$$n = v \times H + h$$

$$n = 0, 1, \dots, N-1$$

$$W_N = e^{\frac{-j2\pi}{N}}$$

Operating Modes

The following sections describe FFT processing types and methods.

Small FFT Computation (<= 256 Points)

A small FFT (NOVER256 = zero) can be handled completely in one step since the twiddles and input data stream fit in the local memories for twiddles and data. In this way two input TCBs (twiddles and data) are fed into the accelerator. After performing the FFT the output TCB writes the results back into the internal memory and the next FFT can start.

Large FFT Computation (>= 512 Points)

For large FFTs (NOVER256 = non zero) the model looks different since the twiddle/data do not fit completely into the local memories. The FFT computation is matrix based on rows (horizontal) and columns (vertical) and performed in three steps:

```
x(0) x(1) x(2) . . . x(H - 1) x(H) x(H + 1) x(H + 2) . . .
x(2H - 1) x(2H) x(2H + 1) x(2H + 2) . . . x(3H - 1) | | | . .
x((V - 1)H) x((V - 1)H + 1) x((V - 1)H + 2) . . . x(VH - 1)
```

1. The vertical (column) V Point FFTs are performed on the matrix.
2. The output of step 1 is multiplied by special twiddles (special coefficients):
3. Horizontal (row) H Point FFTs are performed on the output matrix of Step 2. This produces the final FFT on vertical columns (column wise).

The final FFT result is obtained in internal memory, not local memory.

Example for FFT Size N=512

This example shows a large FFT matrix of $V \times H = 32 \times 16$.

Vertical FFT

1. Input coeff DMA from vertical coeff buffer[64]
2. Input data DMA from input buffer[1024] (modifier = 2H, circbuf = 2N)
3. FFT computation
4. Output DMA to special buffer[1024]

Special Product—Number of Iterations is $N/128 = 4$

1. First iteration:
 - a. Input coeff DMA from special coeff buffer[512]
 - b. Input DMA from special buffer[256]
 - c. FFT computation
 - d. Output DMA to Special buffer[256]
2. Second iteration:
 - a. Input coeff DMA from special coeff buffer[512] (offset = 512)
 - b. Input DMA from special buffer[256] (offset = 256)
 - c. FFT computation
 - d. Output DMA to special buffer[256] (offset = 256)

3. Third iteration:

- a. Input coeff DMA from special coeff buffer[512] (offset = 1024)
- b. Input DMA from special buffer[256] (offset = 512)
- c. FFT computation
- d. Output DMA to special buffer[256] (offset = 512)

4. Fourth iteration:

- a. Input coeff DMA from special coeff buffer[512] (offset = 1536)
- b. Input DMA from special buffer[256] (offset = 768)
- c. FFT computation
- d. Output DMA to special buffer[256] (offset = 768)

Horizontal FFT

1. Input coeff DMA from horizontal coeff buffer[32]
2. Input data DMA from special buffer[1024] (modifier = 2V, circbuf = 2N)
3. FFT computation
4. Output DMA to Output Buffer[1024] (modifier = 2V, circbuf = 2N)

This FFT generates a total of six partial FFT computations. Each computation has an input and output DMA which results in 12 interrupts.

No Repeat Mode

If the FFT_RPT bit = 0, after FFT_START = 1 the accelerator moves from the idle state into the read state (input DMA). After the read completes, the accelerator moves into the processing state then the write state to read the results back into internal memory. The accelerator ends in the idle state.

For large FFTs (based on the VDIM, HDIM and NOVER256 bits) the accelerator knows when the entire FFT processing has finished.

Repeat Mode

If the FFT_RPT bit = 1, after FFT_START = 1 the accelerator moves from the idle state into the read state (input DMA). After the read completes, the accelerator moves into the processing state then the write state to read the results back into internal memory. The accelerator then moves automatically back into the read state for the next FFT frame. In this state multiple linked TCBs which were executed during the first iteration are re-used.

For large FFTs (based on the configuration of the VDIM, HDIM and NOVER256 bits) the accelerator knows when the entire frame processing has finished in order to re-load the new FFT frame parameters at the right time.

Unpacked Data Mode

For small FFTs (FFT<=256), the unpacked data mode can be selected independently for the input or output streams through the use of the FFT_CPACKIN or FFT_CPACKOUT bits (FFTCTL2 register).



The FFT_CPACKIN/FFT_CPACKOUT settings are not applicable for N^3 512 points. The input is always expected to be in alternate real and imaginary format and the output is always generated in the same format.

Inverse FFT

The inverse FFT uses the same algorithm as the forward FFT. The accelerator takes advantage of this fact when processing IFFTs by setting up a coefficient TCB with change of sign for the sine twiddles (FFT uses twiddles cosine, sine, -sine, cosine, the IFFT uses cosine, -sine, sine, cosine). When TCB loading completes, the accelerator processes the inverse FFT and returns the data into the local data memory. Finally, in write mode, data is returned to internal memory.

In order to get the correct amplitude for the inverse FFT, the output buffer needs to be scaled by $1/N$.

Data Transfer

The FFT accelerator works exclusively through DMA and therefore does not require core intervention. This allows the core to perform other system tasks. The core is used to configure the DMA parameter registers and the accelerator control registers and to start accelerator operation.

FFT Buffers

As shown in [Figure 6-1 on page 6-3](#), the input and output DMA stream each pass an 8 deep buffer. These I/O buffers ensure that the FFT stream of the accelerator is not stalled during high DMA bus loads. Note that the buffer status cannot be read.

DMA Transfers

The FFT accelerator supports circular buffer chained DMA. Two TCB structures are associated with input and output DMA. The input TCB structure is used for transferring either data or coefficients to the accelerator block and the output TCB is used for receiving data from the FFT block to the internal memory of the SHARC processor. For TCB structure details see [“FFT Accelerator TCB” on page 2-18](#).

DMA Channels and TCB Structure

The accelerator has two DMA channels that connect to internal memory. The channels fetch the data and coefficients from internal memory and store the results to internal memory. The DMA controller supports circular buffer chain pointer DMA. Separate TCBs must be created for both input and output DMA.

Note that bit 20 of the input chain pointer register (`FFTICP`) indicates whether the TCB is for loading data or coefficients. If the TCB is a coefficient TCB, then circular buffering is not supported and the input length and base registers are ignored.

[Table 2-20 on page 2-18](#) and [Table 2-21 on page 2-18](#) show the input and output TCB structures.

Chained DMA

The DMA controller supports circular buffer chain pointer DMA. The input TCB structure consists of index, modify, count and chain-pointer register values for input data. The input TCB also consists of length and base pointer register values to support circular buffering. Similar to the input TCB structure, the output TCB also consists of index, modify, count, chain pointer, length and base pointer register values to support circular buffered chained DMA for output data.

Once the accelerator is enabled, it loads the TCB values pointed to by the chain pointer register value into its internal registers. The FFT accelerator uses the input TCB values to fetch coefficients and data. It then computes the FFT on the fetched data without any core intervention. Once the computing is complete, the results are stored into the internal memory of the processor using the TCB values of the output TCB registers. If the repeat bit (`FFT_RPT`) is set, the accelerator goes continues on a new FFT frame once the current FFT frame is processed.

One transfer control block (TCB) needs to be configured for each channel. The TCB contains:

- A control register value to configure the FFT parameters for each channel.
- DMA parameter register values for input data.
- DMA parameter register values for twiddles load.
- DMA parameter register values for output data.

Intermediate results for large FFT are stored in the internal memory.



The circular access type is used for large FFTs to process the entire FFT ($V \times H$) matrix.

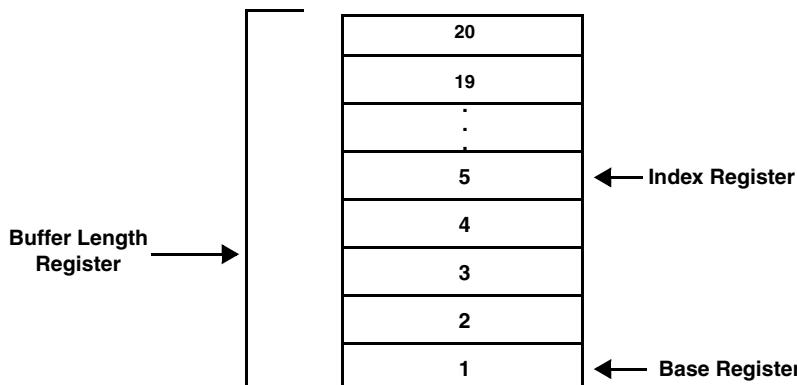


Figure 6-2. Circular Buffer Addressing

Interrupts

The FFT accelerator has two interrupts that are programmable through the programmable interrupt priority control register (see [Appendix B](#),

Peripheral Interrupt Control). Source bits ACC0I and ACC1I are used to connect FFT interrupts to the peripheral interrupt inputs of the core.

Table 6-2 provides an overview of FFT interrupts.

Table 6-2. FFT Interrupt Overview

Interrupt Sources	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
FFT (2 channels)	- Input DMA done - Output DMA done - MAC Status (NaN, Denormal, Underflow, overflow)	Internal transfer completion	RTI instruction	Need to route ACCxI (PICRx) to any PxxI

Interrupt Sources

One interrupt is shared by all the DMA interrupts and the other by MAC status interrupts. Separate status registers are provided to further differentiate the various sources.

Servicing DMA Interrupts

The DMA interrupt is shared by the input DMA and the output DMA. The interrupts are generated at the end of every chain or at the end of an entire DMA sequence, depending on the PCI value in the respective chain pointer registers. When a DMA interrupt occurs, programs can find whether the input DMA interrupt occurred or the output DMA interrupt occurred by reading the DMA status register (FFT_DMSTAT). The DMA interrupt status bits are sticky and are cleared when the DMA status register is read.

Servicing MAC Status Interrupts

A MAC status interrupt is generated whenever a floating-point operation results in an underflow or an overflow condition or when the operands of a floating-point operand is a denormal number or NAN. When a MAC

status interrupt occurs programs can find the cause of the interrupt by polling the MAC status register. The MAC status bits causing the interrupt are sticky and cleared when the MAC status register is read. [For more information, see “Multiplier Status Register \(FTTMACSTAT\)” on page A-78.](#)

FFT Performance

In this section:

V = Number of rows

H = Number of columns

$$N = V \times H$$

- Reads from internal memory take 2 cycles/word.
- Writes to internal memory take 1 cycle/word.
- It takes 2 PCLK cycles to compute a single complex butterfly by the FFT computation.

For performance consideration each FFT computation is accompanied with a preceding Read DMA and a post write DMA.

Small FFT (N is ≤ 256)

Data reads: $2N \times 2$

Butterfly computes: $N \log_2 N$ cycles (A radix2 takes $N/2 \log_2 N \times 2$ PCLK cycles)

Data write: $2N \times 1$

Large FFT ($N \geq 256$)

Total number of performance cycles = (Vertical FFT + Special Prod + Horizontal FFT) cycles.

FFT Accelerator

Vertical FFT cycles

Data and coefficient reads: $2N \times 2 + 2V \times 2$

Butterfly computes: $(V \log_2 V) \times H$

Data writes: $2N \times 1$

Special Prod cycles

Data and coefficient reads: $2N \times 2 + 4N \times 2$

Product compute: $2 \times 4N/4$

Data writes: $2N \times 1$

Horizontal FFT cycles

Data and coefficient reads: $2N \times 2 + 2H \times 2$

Butterfly compute: $(H \log_2 H) \times V$

Data writes: $2N \times 1$

Debug Features

The following sections describe the debugging features available on the accelerator.

Local Memory Access

Setting the `FFT_DBG` bit in the `FFTCTL1` register puts the accelerator into debug mode and allows all memory locations (coefficient and data memory) to be read and written indirectly, using `FTTDADDR` and `FTTDDATA` registers. The MSB bits of the `FTTDADDR` register determines if the access is for the data or the coefficient memory.

Shadow Register

A shadow DMA status register, `FTTSHDMASTAT`, can read the DMA status register without modifying the status values.

Effect Latency

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

Write Effect Latency

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

FFT Accelerator Effect Latency

After the FFT registers are configured the effect latency is 1.5 PCLK cycles minimum and 2 PCLK cycles maximum. Writes to the PMCTL1 register have an effect latency of two PCLK cycles. Wait for at least four CCLK cycles after selecting an accelerator before accessing any of its registers.

Programming Model

There are two separate programming models, one for a FFT that fits in the accelerator's internal memory ($N = 256$ points or less) and one for a FFT that is larger than the accelerator's internal memory ($N = 512$ points or more). In both models, it is assumed that the accelerator starts in idle mode.

$N \leq 256$, No Repeat

For details on the storage format of the coefficients see “[Internal Memory Storage](#)” on page [6-8](#).

1. Configure the ACCSEL bits in the PMCTL1 register to select the FFT accelerator.
2. Program the FFTCTL2 register with:
 $VDIM = N/16$
 $LOG2VDIM = \text{Log}_2(N)$
 $HDIM = 0$

FFT Accelerator

LOG2HDIM = 0
FFT_RPT = 0
FFT_CPACKIN/FFT_CPACKOUT = 0 or 1 depending on whether input/output data is packed into complex words or sent/received data is real or imaginary.

3. Set (=1) the FFT_RST bit in the FFTCTL1 register and wait for a minimum of 4 CCLK cycles.
4. Program control register FFTCTL1 with:
FFT_RST = 0
FFT_EN = 1
FFT_START = 1
FFT_DMAEN = 1
FFT_DEBUG = 0
5. Configure a coefficient DMA to read N complex twiddle factors from the coefficient buffer into the accelerator (total of 2N 32-bit words) and wait until the DMA is complete (or chain DMA in Step 4). This step is not needed if twiddles are already in the coefficient memory of the accelerator.
6. Configure a data DMA to read N complex data points from the input buffer into the accelerator (total of 2N 32-bit words).
7. Configure a data DMA to write N complex data points from the accelerator into the output buffer (total of 2N 32-bit words). There is no need to wait until the DMA in Step 6 completes.
8. Wait until the DMA in Step 7 completes (by interrupt or polling). The computed FFT is now in the core's internal memory and the accelerator is in idle mode.

N <= 256, Repeat

For details on the storage format of the coefficients see “[Internal Memory Storage](#)” on page [6-8](#) .

1. Configure the ACCSEL bits in the PMCTL1 register to select the FFT accelerator.
2. Program the FFTCTL2 register with:
 $V_{DIM} = N/16$
 $\log_2 V_{DIM} = \text{Log2}(N)$
 $H_{DIM} = 0$
 $\log_2 H_{DIM} = 0$
 $\text{FFT_RPT} = 1$
 $\text{FFT_CPACKIN}/\text{FFT_CPACKOUT} = 0$ or 1 depending on whether input/output data is packed into complex words or sent/received data is real or imaginary.
3. Set (=1) the FFT_RST bit in the FFTCTL1 register and wait for a minimum of 4 CCLK cycles.
4. Program the FFTCTL1 register with:
 $\text{FFT_RST} = 0$
 $\text{FFT_EN} = 1$
 $\text{FFT_START} = 1$
 $\text{FFT_DMAEN} = 1$
 $\text{FFT_DEBUG} = 0$
5. Configure a coefficient DMA to read N complex twiddle factors from the coefficient buffer into the accelerator (total of $2N$ 32-bit words) and wait until the DMA is complete (or chain DMA in Step 4). This step is not needed if twiddles are already in the coefficient memory of the accelerator.
6. Configure a data DMA to read N complex data points from the input buffer into the accelerator (total of $2N$ 32-bit words).

7. Configure a data DMA to write N complex data points from the accelerator into the output buffer (total of $2N$ 32-bit words). There is no need to wait until the DMA in Step 6 completes.
8. Wait until the DMA in Step 7 completes (by interrupt or polling). The computed FFTs is now in the core's internal memory and the accelerator is in reading mode, waiting for next batch of FFTs.

N >= 512, No Repeat

For details on the storage format of the coefficients see “[Internal Memory Storage](#)” on page [6-8](#).

Configure the FFT Control Register

1. Configure the ACCSEL bits in the PMCTL1 register to select the FFT accelerator.
2. Factor $N = VH$, where $16 \leq V$ and $16 \leq H$.
3. Set (=1) the FFT_RST bit in the FFTCTL1 register and wait for a minimum of 4 CCLK cycles.
4. Program the FFTCTL2 register with
 $VDIM = V/16$
 $\text{LOG2VDIM} = \text{Log2}(V)$
 $HDIM = H/16$
 $\text{LOG2HDIM} = \text{Log2}(H)$
 $\text{NOVER256} = VH/256$
 $\text{FFT_RPT} = 0$
5. Program the FFTCTL1 register with
 $\text{FFT_RST} = 0$
 $\text{FFT_EN} = 1$
 $\text{FFT_START} = 1$
 $\text{FFT_DMAEN} = 1$
 $\text{FFT_DEBUG} = 0$

Vertical FFT Configuration

6. Configure a coefficient DMA to read 2V twiddle factors from the vertical coeff buffer into the accelerator (total of 2V 32-bit words) and wait until the DMA is complete (or chain DMA in Step 7). This step is not needed if twiddles are already in the coefficient memory of the accelerator.
7. Configure a data transmit DMA to load $2N - 1$ data points from the input buffer into the accelerator with a modify value of 2H, and a circular buffer length of $2N - 1$. Chain a data transmit DMA of count = 1 that loads the last imaginary point.



The `FFT_CPACKIN/FFT_CPACKOUT` settings are not applicable for $N \geq 512$ points. The input is always expected to be in alternate real and imaginary format and the output is always generated in the same format.

8. Configure a data receive DMA to read $2N$ data points from the accelerator into the special buffer with a modify of 1. There is no need to wait until the DMA in Step 6 completes.

Special Buffer Configuration

9. Configure a DMA to load special coefficients from the special coefficients buffer into the accelerator, with a count = 512.
10. Once the DMA in Step 9 completes, configure a data DMA (chained or via interrupt) to read 256 data points (count = 256) from the special buffer into the accelerator with a modify value = 1.
11. Configure a data DMA to write 256 data points (count = 256) from the accelerator into the special buffer with modify value = 1. There is no need to wait until the DMA in Step 9 completes.
12. Repeat step 9 $N/128$ times (offset processing the entire $2N$ buffer of data).

Horizontal FFT Configuration

13. Once the last DMA in Step 10 completes, configure a coefficient DMA to read 2H twiddle factors from the horizontal coeff buffer into the accelerator.
14. Once the DMA in Step 12 completes, configure a data DMA (chained or via interrupt) to read $2N - 1$ data points from special buffer into the accelerator with a modify value = $2V$ and a circular buffer length of $2N - 1$. Chain a data DMA of count = 1 that reads the last imaginary point.
15. Configure a data DMA to write $2N - 1$ data points from the accelerator into the output buffer with a modify value = $2V$ and a circular buffer length of $2N - 1$. There is no need to wait until the DMA in Step 9 completes. Chain a data DMA of count = 1 that reads the last imaginary point.
16. Wait until the DMA in step 14 completes (by interrupt or polling). The computed FFT is now in the output buffer and the accelerator is in idle mode.

N >= 512, Repeat

For details on the storage format of the coefficients see “[Internal Memory Storage](#)” on page [6-8](#).



Transmit DMAs take place using input TCBs; receive DMAs take place using output TCBs.

1. Configure the ACCSEL bits in the PMCTL1 register to select the FFT accelerator.
2. Factor $N = VH$, where $16 \leq V$ and $16 \leq H$.
3. Set (=1) the FFT_RST bit in the FFTCTL1 register and wait for a minimum of 4 CCLK cycles.

4. Program the FFTCTL2 register with:

$V_{DIM} = V/16$

$\log_2 V_{DIM} = \text{Log2}(V)$

$H_{DIM} = H/16$

$\log_2 H_{DIM} = \text{Log2}(H)$

$N_{OVER256} = VH/256$

$\text{FFT_RPT} = 1.$

5. Program the FFTCTL1 register with:
 $\text{FFT_RST} = 0$

$\text{FFT_EN} = 1$

$\text{FFT_START} = 1$

$\text{FFT_DMAEN} = 1$

$\text{FFT_DEBUG} = 0$

For steps 6–15, see “[N >= 512, No Repeat](#)” above.

Debug Mode

The next sections show the steps required for reading and writing local memory in debug mode.

Write to Local Memory

1. Enable the FFT module using the PMCTL1 register.
2. Wait at least 4 CCLK cycles.
3. Clear the FFT_DMAEN bit in the FFTCTL1 register.
4. Set the FFT_DBG bit in the FFTCTL1 register.
5. Write first data to the FFTDDATA register.
6. Write address to the FFTDADDR register. Note the MSB Address bits determines which memory to write.
7. Wait at least 12 CCLK cycles before writing again FFTDDATA register.

Read from Local Memory

1. Enable FFT module using the PMCTL1 register.
2. Wait at least 4 CCLK cycles.
3. Clear the FFT_DMAEN bit in the FFTCTL1 register.
4. Set the FFT_DBG bit in the FFTCTL1 register.
5. Write address to the FFTDADDR register. The MSB address bits determine which memory to read.
6. Wait at least 20 CCLK cycles before writing data to FFTDATA register.

FIR Accelerator

Finite Impulse Response (FIR) filters are used in a wide array of applications, and can be used in multi-rate processing in conjunction with an interpolator or decimator.

Features

This hardware module is capable of performing FIR filters without core intervention. This gives programs freedom to use the core to implement complex algorithms, effectively adding more bandwidth to the processor.

- FIR supports fixed point and IEEE floating point format
- Single rate or multi-rate window processing
- Change the rates with decimation or interpolation mode
- Up to 32 filter channels available

Register Overview

The FIR accelerator registers are described below.

Power Management Control Register (PMCTL1). Used for FIR accelerator selection. Controls the clock power down to the module if not required.

Control Registers (FIRCTLx). Used to configure the global parameters for the accelerator. These include the number of channels, channel auto iterate, DMA enable, and accelerator enable.

The `FIRCTL2` register is used to configure the channel specific parameters such as filter TAP length, window size, sample rate conversion, up/down sampling and ratio.

DMA Status Register (FIRDMASTAT). Provides the status of the FIR accelerator operation. This information includes chain pointer loading, coefficient DMA, data preload DMA, processing in progress, window processing complete, and all channels processing complete.

MAC Status Register (FIRMACSTAT). Provides the status of MAC operation for all four multiply accumulators. In fixed-point mode, only the ARI_X (adder result infinity) is used, all other bits are reserved.

Debug Control Register (FIRDEBUGCTL). Controls the debug mode operation of the accelerator.

Clocking

The FIR accelerator runs at the maximum speed of the peripheral clock frequency (f_{PCLK}).

Functional Description

Figure 6-3 shows the block diagram of the 1024-TAP FIR hardware accelerator. The accelerator consists of a 1024 word coefficient memory, a 1024 deep delay line for data, and four MAC units. The accelerator runs at the peripheral clock frequency (PCLK).

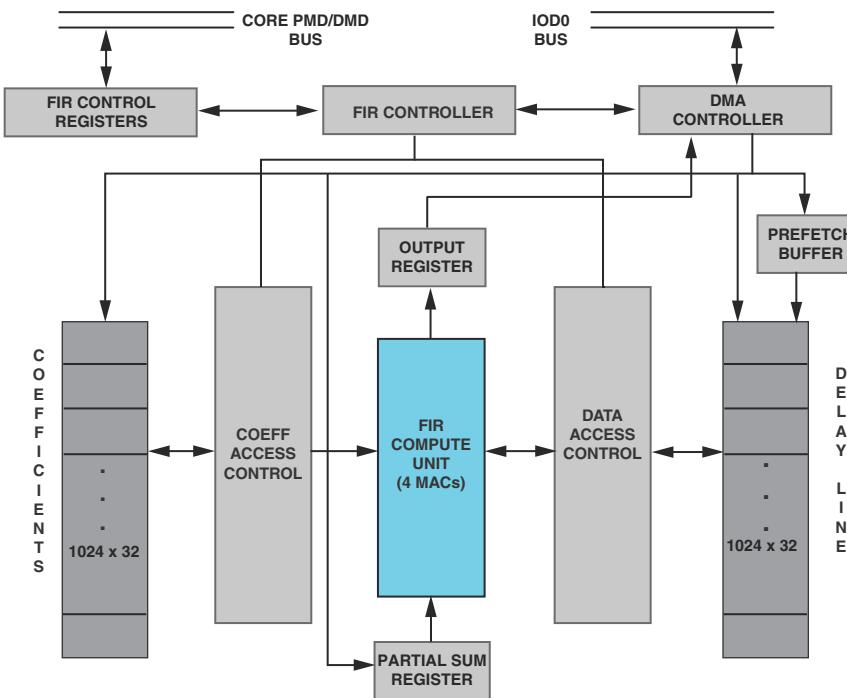


Figure 6-3. FIR Block Diagram

The FIR accelerator has following logical sub blocks.

1. A data path unit that consists of:
 - a. A 1024 deep coefficient memory
 - b. A 1024 deep delay line for the data

- c. Four 32-bit floating-point and fixed-point multiplier and adder units
 - d. One 32-bit buffer to efficiently supply data to the data path
 - e. One 32-bit buffer to hold previous partial sum
 - f. One 32-bit buffer to hold the output
2. Configuration registers for the number of TAPs, number of channels, filter enable, interrupt control, DMA enable, up sample/down sample control, and ratios.
 3. Core access interface for writing the DMA/filter configuration registers and reading the status register.
 4. DMA bus interface for transferring data and/or coefficients to and from the accelerator.
 5. DMA configuration registers including chain pointer, input, output, and coefficient registers.

Compute Block

The MAC unit, shown in [Figure 6-4](#), has four multiply accumulators. They operate simultaneously on a single filter as described below.

- The MAC unit operates on the data and coefficient fetched from the data and coefficient RAMs.
- Each MAC can perform 32-bit floating-point or 32-bit fixed-point MAC operations.
- Floating-point format is IEEE compliant.
- Multiply and accumulation operation (addition) are pipelined.

- 32-bit floating-point MAC operation generates 32-bit multiply results.
- 32-bit fixed-point operation generates 80-bit results.

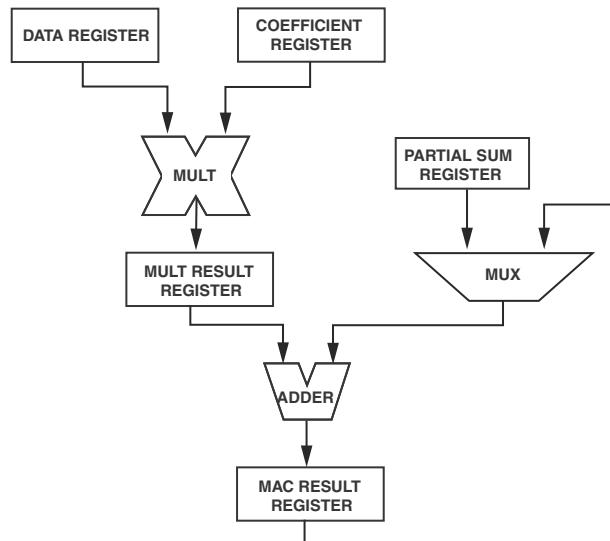


Figure 6-4. FIR MAC Unit

Partial Sum Register

The partial sum register is useful for floating-point multi-iteration mode. For a particular channel, the intermediate MAC result is written to the internal memory's output buffer. If the same channel is requested again, the partial result register is updated with the intermediate MAC result via DMA from the internal memory's output buffer and added to the current MAC result after each iteration. This process is repeated until all iterations are done (the entire soft filter length is processed).

Delay Line Memory

The accelerator has a 1024 TAP delay line to hold the data locally. The DMA controller fetches the data from internal memory and loads it into the delay line. Four read accesses can be made to the delay line simultaneously.

Coefficient Memory

The accelerator has a 1024 deep coefficient memory to store the coefficients. The DMA controller loads the coefficients from internal memory into coefficient memory. Four coefficients can be fetched from the coefficient memory simultaneously. If the soft filter length is more than 1024, processing is done in multi-iteration mode.

Prefetch Data Buffer

This buffer is used to pre-fetch and keep the next input sample from the memory (in parallel), when the compute unit is operating on the delay-line corresponding to the current sample. The data pre-fetched in this buffer is later used to update the delay line for the next sample. This happens in parallel again, when the compute unit is not accessing the delay line in other words when it is adding the output from the four MACs and the partial sum register.

Processing Output

The accelerator uses all four MACs simultaneously to calculate one output sample as shown in [Figure 6-5](#) and the following procedure.

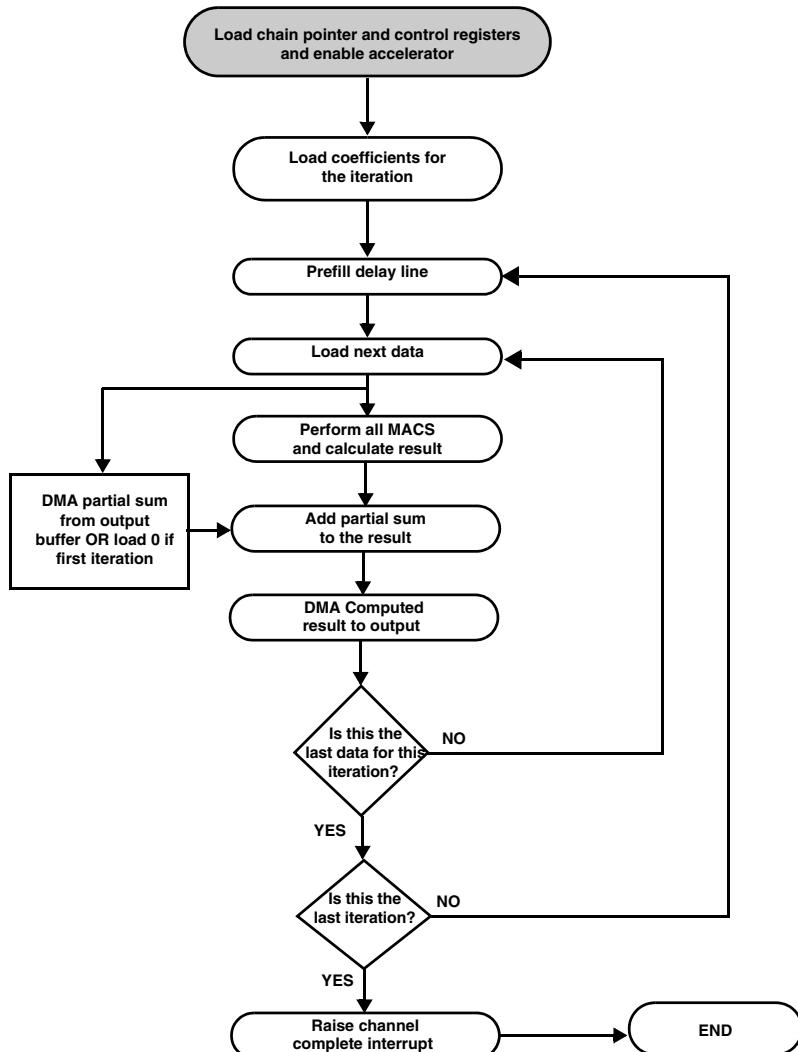


Figure 6-5. Multi-Iteration Filtering Flow

1. The accelerator fetches four input data from the delay line and four corresponding coefficients from the coefficient memory and feeds them to the MAC units for multiply/accumulation.
2. The accelerator repeats the procedure with the next four input data and coefficients until all the TAPs complete. For an N TAP filter for example, this procedure is done $N/4$ times.
3. When all the TAPs complete, the accelerator adds the four MAC outputs together to the previous partial sum (if any) to calculate the final result.
4. Finally, that output sample is stored back in internal memory.

Internal Memory Storage

The following sections describe the storage format for the accelerator.

Coefficients and Input Buffer Storage

For any N TAP filter with coefficients:

$C[i]$ $i = 0, 1,$

...

$N - 1$

the coefficients should be stored in internal memory buffer in the order:

$C[N - 1], C[N - 2]$

...

$C[1], C[0]$

and the CI should point to $C(N - 1)$

Single Rate Input Filtering

The total size of the input buffer should at least be equal to $N - 1 + W$. If the input buffer that needs to be processed is:

$x[n], x[n+1], x[n+2]$

....

$x[n+W-1]$

it should be stored in the memory as

$x[n-(N-1)], x[n-(N-2)]$

....

$x[n-1], x[n], x[n+1]$

....

$x[n+W-1]$

and IIFIR should point to $x[n - (N - 1)]$

Decimation

Assuming M = decimation ratio, the total size of the input buffer should at least be equal to $N-1+W\times M$. If the input buffer that needs to be processed is

$x[n], x[n+1], x[n+2] \dots x[n+W\times M-1]$,

it should be stored in the memory as

$x[n-(N-1)], x[n-(N-2)] \dots x[n-1], x[n], x[n+1] \dots x[n+W\times M-1]$

and IIFIR should point to $x[n-(N-1)]$.

Interpolation

Assuming L = interpolation ratio, the total size of the input buffer should at least be equal to $\text{Ceil}((N-1)/L)+W/L$.

If the input buffer that needs to be processed is

$x[n], x[n+1], x[n+2] \dots x[n+W/L-1]$, and $K = \text{Ceil}((N-1)/L)$

it should be stored in the memory as:

$x[n-(K-1)], x[n-(K-2)] \dots x[n-1], x[n], x[n+1] \dots x[n+W/L-1]$

and the IIFIR should point to $x[n-(K-1)]$.

Operating Modes

The FIR core performs a sum-of-products operation to compute the convolution sum. It supports single-rate, decimation, and interpolation functions.

Single Rate Processing

In a single-rate filter, the output result rate is equal to the input sample rate. The filter output $Y(n)$ is computed according to following equation where N is the number of filter coefficients: $c(i)$ $i = 0, \dots, N - 1$ are the filter coefficients and $x(n)$ represents the input time-series.

$$Y(n) = \sum_{k=0}^{N-1} c(k) \times x(n-k)$$

Single Iteration

Results are computed in single iteration when the soft filter length is less than or equal to 1024.

Multi-Iteration

Results are computed in multiple iterations when the soft filter length is greater than 1024 (for example, 2048 TAPs on a 1024 hard filter length). In this mode, the controller implements two iterations of 1024 TAPs. Note that if the soft filter length is not a multiple of the hard filter length the controller does iterate until the soft filter length is satisfied.

Example: 550 taps on a 256 tap filter.

In this example, the FIR controller implements two iterations of 256 taps and one iteration of 38 taps.



Multi-iteration mode is not supported in fixed-point format.

Window Processing

Sample based processing mode is selected by configuring window size to 1. In this mode, one sample from a particular channel is processed through all the biquads of that channel and the final output sample is calculated.

In window based mode, multiple output samples (up to 1024) equal to the window size of that channel are calculated. After these calculations are complete, the accelerator begins processing the next channel. A configurable window size parameter is provided to specify the length of the window.

Multi Rate Processing

Multi rate filters change the sampling rate of a signal—they convert the input samples of a signal to a different set of data that represents the same signal sampled at a different rate.

Decimation

A decimation filter provides a single output result for every M input samples, where M is the decimation ratio. Note that the output rate is 1/M'th of the input rate. The filter implementation exploits the low output sample rate by not starting a computation until a new set of M input samples is available.

In this mode, after low pass filtering (for anti aliasing), FIR logic discards the ratio – 1 samples of output data. For performance optimization, FIR logic skips the computation of output samples, which are discarded.

The input buffer size for decimation filters is $N - 1 + (W \times M)$ where:

- N is the number of taps
- W is the window size
- M is the decimation ratio

The window size (WINDOW bits) in the `FIRCTL2` register must be programmed with the number of output samples.

To start this mode, programs set the `FIR_RATIO` and `FIR_UPSAMP` bits in the `FIRCTL2` register (along with normal filter setting). Also the `TAPLEN` bits setting should be greater than or equal to `FIR_RATIO` bits setting for decimation filter.

Interpolation

An interpolator filter provides L output results for each new input sample, where L is the interpolation ratio. Note that the output rate is L times the input rate.

In this mode, according to the ratio specified in configuration register, FIR logic inserts L – 1 zeros between any two input samples (up-sampling) and then performs the interpolation (through the FIR filter).

Both up-sampling and down-sampling do not support multi iteration mode. Therefore, the filtering operation can only be done on up to 1024 TAPs and the ratio of up/down sampling can only be an integer value.

In an interpolation filter FIR logic inserts L – 1 zeros between each sample and the program has to make sure that these zeroes are fully shifted out of the delay line before moving on to the next channel. This puts a restriction on window size in terms of L – *the sample ratio* as shown below.

$$\text{WINDOWSIZE} = n \times \text{SAMPLERATIO}$$

where n is the number of input samples.

The input buffer size is the smallest integer greater than or equal to $(N - 1 + W)/L$ for interpolation filters where:

- N is the number of taps
- W is the window size
- L is the interpolation ratio

To start the mode, programs configure the `FIR_RATIO` and `FIR_UPSAMP` bits (along with filter settings) in the `FIRCTL2` register.

Channel Processing

[Figure 6-6 on page 6-41](#) shows the flow diagram for processing a single channel. Channels are processed in TDM format by setting the `FIR_CH` bits greater one. In the time slot corresponding to a particular channel, the corresponding TCB is loaded from internal memory.

1. The `FIRCTL2` value is fetched from internal memory and is used to configure the filter parameters for that channel.
2. The accelerator fetches the coefficients using the `CIFIR` register as the pointer and loads them into coefficient memory.
3. The delay line is pre-filled using the `IIFIR` register as the pointer.
4. The accelerator calculates the first output and stores the result back into the output buffer using the `OIFIR` register as the pointer.
5. While calculating the output the accelerator fetches the next data in parallel. After one window of data is processed, the index registers in the internal memory TCB are updated so that in the next time slot of the same channel, processing can be continued from where it stopped.

6. Processing moves to the next channel and repeats the procedure. If the soft filter length is more than the hard filter length, multiple iterations are done to process the window.

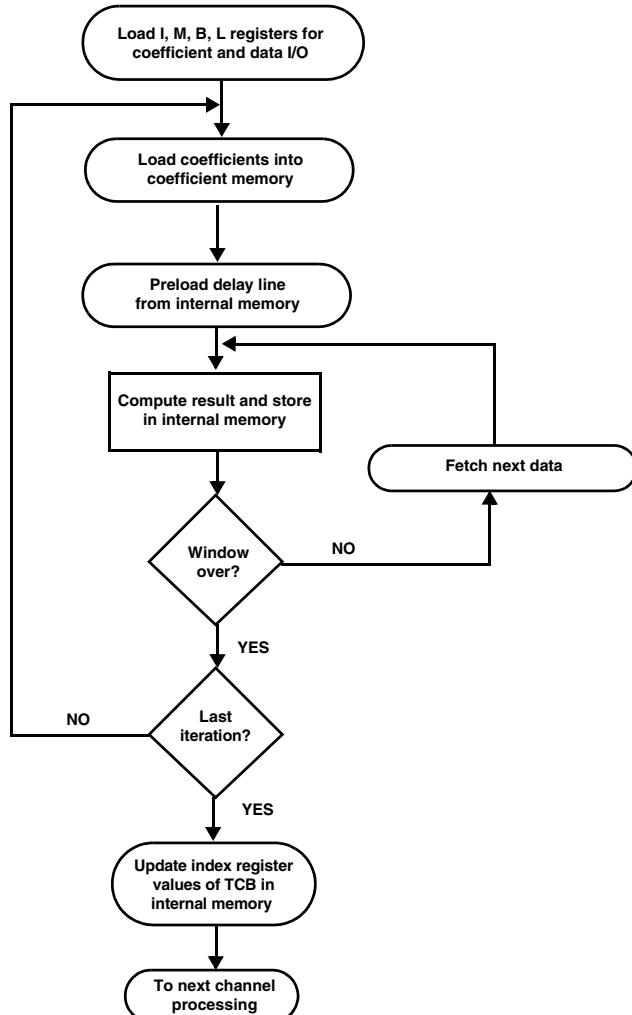


Figure 6-6. Single Channel Filtering Flow

Floating-Point Data Format

The FIR accelerator treats data and coefficients in 32-bit floating-point format as the default functional mode.

Fixed-Point Data Format

In fixed-point mode, the 32-bit input data/coefficient is treated as fixed-point. A 32-bit fixed-point MAC operation generates an 80-bit result. Fixed-point data/coefficients can be unsigned integer, unsigned fractional and signed integer.

-  In fixed point mode, the entire 80-bit result register is always written back in bursts of 3×32 bits. The first word is the LSW, the 2nd the MSW and the third word is a 16-bit overflow, the remaining 16-bits are padded with zeros. Therefore for fixed-point $\text{WINDOWSIZE} = \text{WINDOWSIZE} \times 3$.

If signed fractional format is used, the output needs to be scaled by 2 since the MAC does not the right shift to remove the redundant sign bit. A final routine needs to decimate the output buffer to the desired samples.

Multi iteration mode is not supported in this format. Therefore, the maximum TAP length is 1024.

Data Transfer

The FIR filter works exclusively through DMA.

DMA Access

The FIR accelerator has two DMA channels (accelerator input and output) to connect to the internal memory. The DMA controller fetches the data and coefficients from memory and stores the result.

Chain Pointer DMA

The DMA controller supports circular buffer chain pointer DMA. One transfer control block (TCB) needs to be configured for each channel. The TCB contains:

- A control register value to configure the filter parameters for each channel
- DMA parameter register values for the input data (delay line)
- DMA parameter register values for coefficient load
- DMA parameter register values for output data
- Intermediate results in multi-iteration mode are saved in the output buffer

As shown in “[FIR Accelerator TCB](#)” on page 2-16 and [Figure 6-7](#), the accelerator loads the TCB into its internal registers and uses these values to fetch coefficients and data and to store results. After processing a window of data for any channel, the accelerator writes back the appropriate values to the `IIFIR` and `OIFIR` fields of the TCB in memory, so that data processing can begin from where it left off during the next time slot of that channel.

The writeback value for input buffer is:

- $IIFIR + W$ for single rate filtering.
- $IIFIR + W \times M$ for decimation (M = decimation ratio).
- $IIFIR + W/L$ for interpolation (L = interpolation ratio).
- The writeback values for output buffer in floating point mode is: $OIFIR + W$.
- The writeback values for output buffer in fixed point mode is: $OIFIR + 3 \times W$.

i The FIRCTL2 register is part of the FIR TCB. This allows programming individual FIR channels with different control attributes.

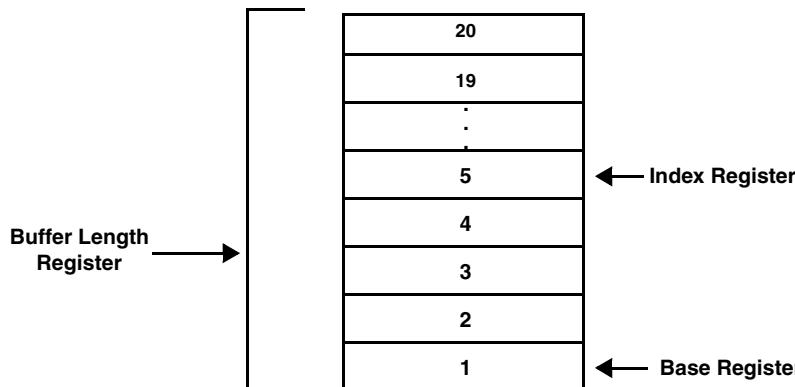


Figure 6-7. Circular Buffer Addressing

Interrupts

The FIR accelerator has two interrupts (Table 6-3) that are programmable through the programmable interrupt priority control register (see [Appendix B, Peripheral Interrupt Control](#)). Source bits ACC0I and ACC1I are used to connect FIR interrupts to the peripheral interrupt inputs of the core.

Table 6-3. Overview of FIR Interrupts

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
FIR (2 channels)	- Window Complete - Channel Complete - MAC status	- internal transfer completion	RTI instruction	Need to route ACCx1 (PICRx) to any PxxI

One interrupt line is shared by all the DMA interrupts and the other by MAC status interrupts. Separate status registers are provided to further differentiate the various sources.

Interrupt Sources

There are two types of DMA interrupt sources associated with the accelerator. The `FIR_CCINTR` bit in the `FIRCTL1` register controls these interrupts.

Window Complete Interrupt – This interrupt is generated at the end of each channel when all the output samples are calculated corresponding to a window and updated index values are written back.

All Channels Complete Interrupt – This interrupt is generated when all the channels are complete or when one iteration of time slots completes.

MAC Status Interrupt – The status interrupt sources are derived from the `FIRMACSTAT` register. For more information, see “[FIR MAC Status Register \(FIRMACSTAT\)](#)” on page A-83.

Service Channel Interrupts – Based on the `FIR_CCINTR` bit in the `FIRCTL1` register both bits (`FIR_DMAWDONE` or `FIR_DMACADONE`) are set in the `FIRDMSTAT` register if either of the conditions is met. The interrupt service routine should read (to clear) both bits.

Service MAC Status Interrupts – A MAC status interrupt is generated whenever a fixed or floating-point operation results in an arithmetic exception. Reading the `FIRMACSTAT` register returns for which MAC unit is causing an exception.

Debug Features

The following sections provide information of debugging the FIR accelerator.

Local Memory Access

The contents of FIR delay line and coefficient memories are made observable for debug by setting the `FIR_DBGMODE`/`FIR_DBGMEM` and `FIR_HLD` bits in the `FIRDEBUGCTL` control register. The debug address register (`FIR_DBGADDR`) and two data registers are provided for debug operations.

Bit 11 of the `DBGADDR` register selects coefficient memory if set (=1) and selects delay line memory in cleared (=0).

In the debug mode, the read data register (`DBGMEMRDDAT`) returns the contents of the memory location pointed to by the address register. Data can be written into any memory location using `DBGMEMWRDAT` register writes. If the address auto increment bit (`FIR_ADRINC`) is set, the address register auto increments on `DBGMEMWRDAT` writes and `DBGMEMRDDAT` reads. During auto increment, the `FIR_DBGADDR` register cannot cross the data memory/coefficient memory boundary.

Single Step Mode

Programs can single step through the MAC operations and observe the memory contents after each step. The `FIR_DBGMODE/FIR_HLD` and `FIR_RUN` bits control the FIR MAC units.

Emulation Considerations

In FIR debug mode, the DMA operations are not observable.

Effect Latency

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

Write Effect Latency

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

FIR Accelerator Effect Latency

After the FIR registers are configured the effect latency is 1.5 `PCLK` cycles minimum and 2 `PCLK` cycles maximum. Writes to the `PMCTL1` register have

an effect latency of two PCLK cycles. Wait for at least four CCLK cycles after selecting an accelerator before accessing any of its registers.

FIR Throughput

Accelerator input and output channels are used to connect to internal memory. Data throughput is one 32-bit data word per peripheral clock cycle for writes to memory, provided there are no conflicts. Read throughput from memory, throughput is one 32-bit data word per two peripheral clock cycles.

The following information describes the performance of the FIR accelerator in processor cycles.

Total number of PCLK cycles for single rate filtering $N \leq 1024$ is:

$$(\text{TCB load} + 4 \times N + W(N/4 + 2)) \times C$$

and the total number of PCLK cycles for desimation is:

$$(\text{TCB load} + 4 \times N + W(N/4 + 2) + (W - 1) \times (M - 1) \times 7) \times C \text{ where:}$$

- N – Number of taps
- W – Window size
- C – Number of channels
- TCB load = 49 PCLK cycles
- $4 \times N$ – Number of cycles for loading coefficients and data considering two cycles for read
- $N/4 + 2$ – FIR compute cycles considering four pipelined MACs
- M – Decimation ratio

Additional Information

It may be difficult to achieve the required performance by using sample based processing (Window size = 1). Increasing the window size provides more computation time and the ability to perform the real time processing. It is a common practice to use block based processing (Window size greater than 1).

In the following example, the core is running at 450 MHz with a single channel tap length of 512. Sampling frequency = 96 kHz and the maximum processing time is 10 μ s.

The computation of the filter requires:

$$\text{TCB load} + 4 \times N + W(N/4 + 2) \times C.$$

For window size = 1

$$49 + 4 \times 512 + 1 \times (512/4 + 2) = 2227 \text{ PCLK} = 9.8 \text{ us.}$$

For window size = 10

$$49 + 4 \times 512 + 10 \times (512/4 + 2) = 3397 \text{ PCLK} = 15 \text{ us.}$$

Therefore, 10 samples at 96 kHz = $10 \times 10 = 100$ us of available processing time: Actual time used is 15 us.

Programming Model

The following steps should be used when programming the accelerator. Enable the FIR accelerator by setting accelerator select bits (`ACCSEL` in the `PMCTL1` register) to 00.

Single Channel Processing

1. Create input, coefficient, and output buffers in internal memory.

For input and coefficient buffer storage format see “[Coefficients and Input Buffer Storage](#)” on page [6-35](#).

2. Create the TCBs in internal memory. Each TCB corresponds to a particular channel.

TCBs hold the `FIRCTL2` register which allows programming the window size and tap size along with up or down sample enable, sample rate conversion enable, and the conversion ratio for decimation and interpolation filters.

3. Configure the index, modifier, length entries in the TCBs to point to the corresponding channels' data buffer, coefficient buffer, and output data buffer.

The output index register should always point to the start of the output buffer. However, the input index register's value should be initialized based on the explanation provided in “[Coefficients and Input Buffer Storage](#)” on page 6-35.

4. The core configures the `FIRCTL1` register with the number of channels (one channel), fixed- or floating-point format.
5. Set the enable bit to start accelerator operation in the modes configured (in `FIRCTL1` and `FIRCTL2` registers) by loading the first channels' TCB. Once the first channel window is calculated, the input and output index registers are written back to internal memory corresponding to the first channel. Once the write back is complete the accelerator moves into idle.

Multichannel Processing

[Figure 6-8 on page 6-51](#) shows the diagram for multichannel filtering. Multiple channels are processed in a time division multiplexed (TDM) format. After completing all the channels, the accelerator can either repeat the slots or wait for core intervention.

For multichannel filtering, use the following steps.

1. Program the number of channels in the `FIRCTL1` register using the `FIR_NCH` bits (5–1).
2. Configure the TCBs in internal memory with one channel's TCB pointing to the next channel's TCB.
3. Write the first TCB value into the `CPFIR` register and enable the accelerator.

The accelerator fetches the first channel's TCB and, using it as pointer, pre-fills the delay line and coefficient memory and loads the `FIRCTL2` register to configure the filter parameters corresponding to that channel.

The accelerator then calculates output samples corresponding to one Window and stores the data back in internal memory.

At the end of the Window the accelerator updates the `IIFIR` and `OIFIR` registers in the TCB of internal memory and moves to the next channel.

When all the channels are finished and the auto channel iterate (`CAI`, bit 9) is set, the accelerator processes the first channel again and iterates through the channels. If the `CAI` bit is cleared, the accelerator waits for core intervention.

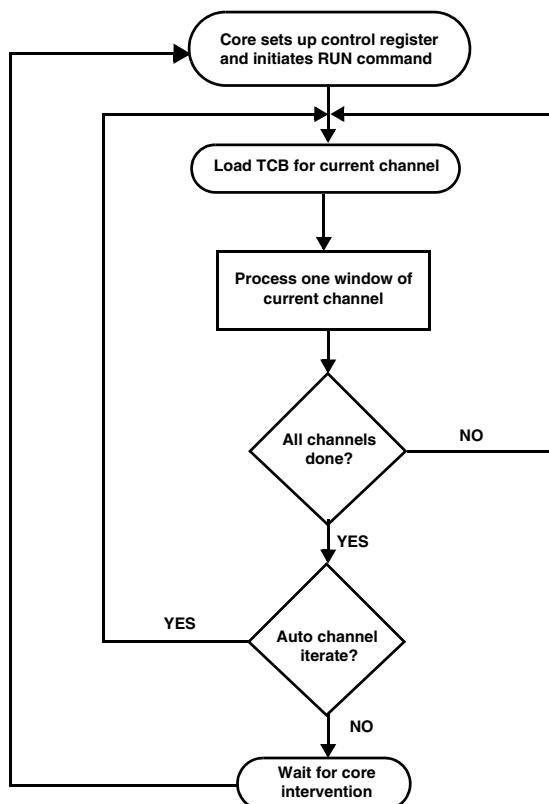


Figure 6-8. Wait for Core Intervention => Idle (if CAI bit = 0)

Debug Mode

The next sections show the steps required for reading and writing local memory in debug mode.

Write to Local Memory

1. Enable the FIR module using the `PMCTL1` register.
2. Wait at least 4 `CCLK` cycles.
3. Clear the `FIR_DMAEN` bit in the `FIRCTL1` register.
4. Set `FIR_DBGMODE`, `FIR_DBGMEM` and `FIR_HLD` bits in `FIRDEBUGCTL` register.
5. Set the `FIR_ADRINC` bit in `FIRDEBUGCTL` register for address auto increment.
6. Write start address to the `FIRDBGADDR` register. Note if bit 11 is set, coefficient memory is selected.
7. Wait at least 4 `CCLK` cycles.
8. Write data to the `FIRDBGWRDATA` register.

Read from Local Memory

1. Enable FIR module using the `PMCTL1` register.
2. Wait at least 4 `CCLK` cycles.
3. Clear the `FIR_DMAEN` bit in the `FIRCTL1` register.
4. Set `FIR_DBGMODE`, `FIR_DBGMEM` and `FIR_HLD` bits in `FIRDEBUGCTL` register.
5. Set the `FIR_ADRINC` bit in `FIRDEBUGCTL` register for address auto increment.

6. Write start address to the `FIRDBGADDR` register. Note if bit 11 is set, coefficient memory is selected.
7. Wait at least 4 `CCLK` cycles.
8. Read data from the `FIRDBGRRDATA` register.

Single Step Mode

Single step mode can be used for debug purposes. An additional debug register is used in this mode.

1. Enable stop DMA during breakpoint hit in the emulator settings.
2. Clear the `FIR_HLD` bit and enable `FIR_DBGMODE` and `FIR_RUN` bits in `FIRDEBUGCTL` register.
3. Program FIR module according to the application.
4. In single step each iteration is updated in the emulator session.

FIR Programming Example

An application needs FIR filtering of six channels of data. The first four channels require 256 TAP filtering and the last two channels require 1024 TAP filtering. The window size for all the channels is 128.

1. Create a circular data buffer in internal memory for each channel.

The buffer should be large enough to avoid overwriting data before being processed by the accelerator. Ideally, the input buffer size for a channel is *tap length + window size – 1* for that channel. The 256 coefficients of each of the first four channels and the 1024 coefficients each of the last two channels are also configured in internal memory buffers. The output buffer size is equal to the window size.

2. Create six TCBs in internal memory with each channel's chain pointer (CP) entry pointing to the next channel's and the sixth channel's CP entry pointing back to the first's in a circular fashion.
3. Configure the `FIRCTL2` register for the first four channels' TCBs to 256 TAPs and a window size of 128, and the next two channels for 1024 TAPs and a window size of 128, respectively.
4. Configure the index, modifier, length entries in the TCBs to point to the corresponding channel's data buffer, coefficient buffer, and output data buffer. The location of the first channel's TCB is written to the `CPFIR` register. The `FIRCTL1` register is then programmed with an `FIR_CH` value that corresponds to six channels.
 - a. The accelerator iterates through six channels once and then waits for core intervention, (the `FIR_CAI` bit is not set, the DMA is enabled, and the `FIR_EN` bit is set).
 - b. The accelerator then loads the first channel's TCB then loads the coefficient and data and processes one window.
 - c. After saving the index values to memory the accelerator moves to the next channel.
 - d. After all six channels are complete the accelerator halts and waits for core intervention.

IIR Accelerator

The ADSP-214xx processors have an IIR filter accelerator implemented in hardware, that reduces the processing load on the core, freeing it up for other tasks.

Features

The accelerator supports a maximum of 24 channels. There is support for up to 12 cascaded bi-quads per channel. This means that the accelerator locally stores all the biquad coefficients of 24 channels. Window size can be configured from 1 (sample based) to 1024.

- IIR supports IEEE floating point format 32/40-bit
- Sample based or window based processing
- Up to 12 cascaded biquads per channel
- Up to 24 filter channels available

Register Overview

The following sections provide information on the IIR accelerator control and status registers.

Power Management Control Register (PMCTL1). Used for IIR accelerator selection. Controls the clock power down to the module if not required.

IIR Global Control (IIRCTLx). Configures the global parameters for the accelerator. These include number of channels, channel auto iterate, DMA enable, and accelerator enable.

The `IIRCTL2` register is used to configure the channel specific parameters. These include number of biquads and window size.

DMA Status (IIRDMASTAT). Provides the status of accelerator operation including chain pointer loading, coefficient DMA, processing progress, window complete and all channels complete.

MAC Status (IIRMACSTAT). Provides the status of the MAC operations.

Debug Mode Control (IIRDEBUGCTL). Controls the debug mode operation of the accelerator.

Clocking

The IIR accelerator runs at the maximum speed of the peripheral clock (f_{PCLK}).

Functional Description

[Figure 6-9](#) shows the block diagram of the IIR hardware accelerator. The accelerator has a coefficient memory size of 1440 x 40 bits, a data memory size of 576 x 40 bits and one MAC unit with an input data buffer to supply data to the MAC.

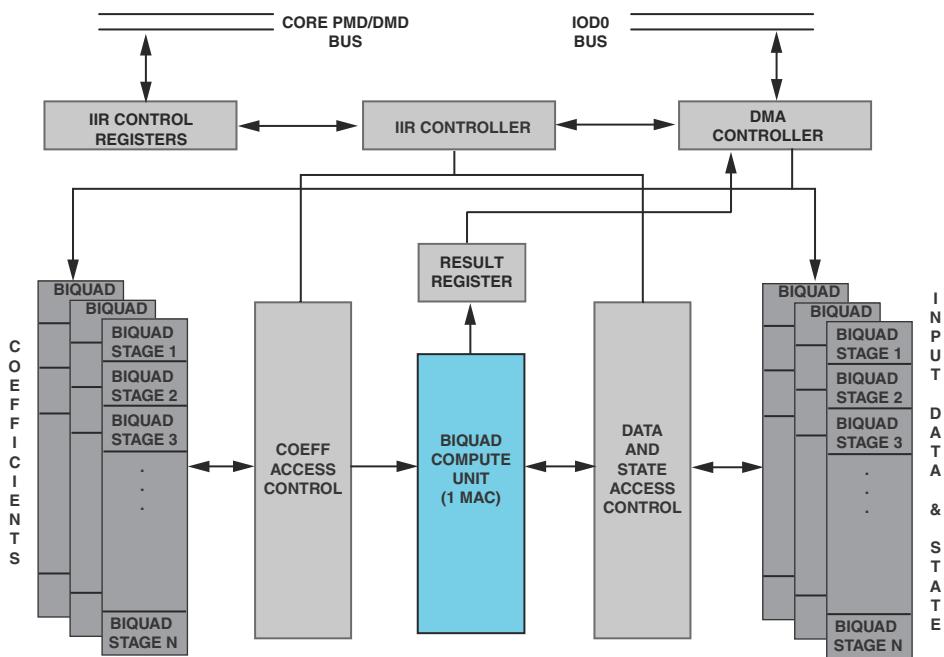


Figure 6-9. IIR Accelerator Block Diagram

The IIR accelerator is implemented using Transposed Direct Form II biquad which has less coefficient sensitivity. [Figure 6-10](#) shows the signal flow graph for the biquad structure.

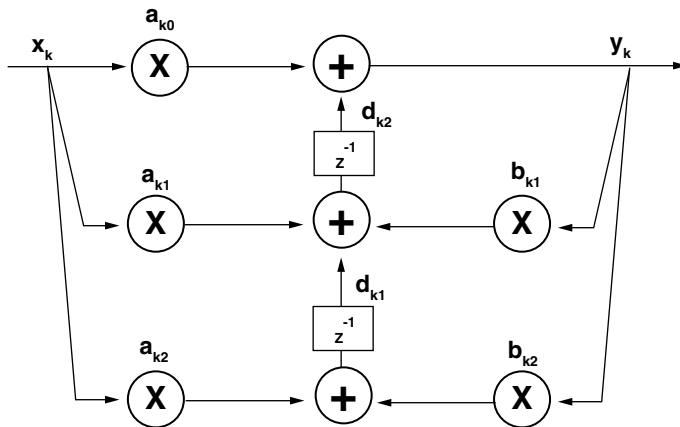


Figure 6-10. Transposed Direct Form II Biquad

The accelerator has the following logical sub blocks:

- A data path unit with the following elements:
 - 32/40-bit coefficient memory for storing biquad coefficients
 - 32/40-bit data memory for the intermediate data
 - One 40/32-bit floating-point multiplier and adder (MAC) unit
 - An input data buffer to efficiently supply data to MAC
 - One 40-bit result register to hold result of biquad
- Configuration registers for controlling various parameters such as the number of biquads, the number of channels, interrupt control, and DMA control
- A core access interface for writing the DMA/filter configuration registers and for reading the status registers

- A DMA bus interface for transferring data to and from the accelerator. This interface is also used to preload the coefficients and Dks at start up.
- DMA configuration registers for the transfer of input data, output data and coefficients

Multiply and Accumulate (MAC) Unit

The MAC unit shown in [Figure 6-11](#) has a pipelined multiplier and accumulator unit that operates on the data and coefficient fetched from the data and coefficient memory. The MAC can perform either 32-bit floating-point or 40-bit floating-point MAC operations. 32-bit floating-point operations generate 32-bit results and 40-bit floating-point operations generate 40-bit results.

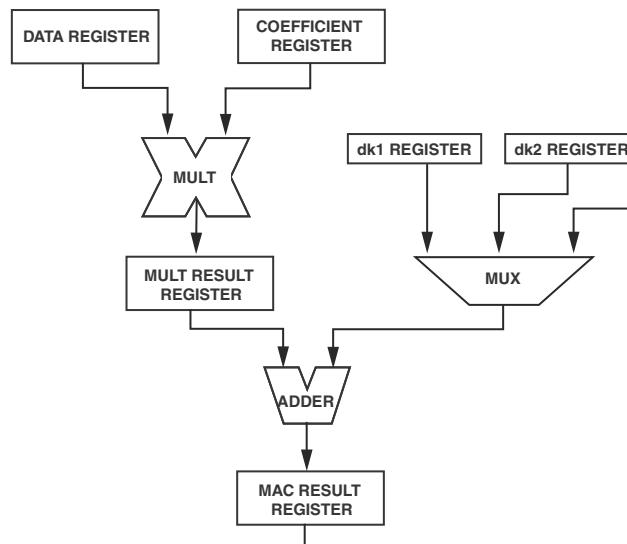


Figure 6-11. IIR MAC Unit

Data Memory

The size of data memory is 576 x 40 bits and is used to hold the dk1 and dk2 intermediate data of all the biquads locally. The DMA controller fetches the sample data from internal memory and calculates the output as well as the dk1 and dk2 values for each biquad and stores them in data memory.

Coefficient Memory

The size of coefficient memory is 1440 x 40 bits and is used to store all the coefficients of all the biquads. At start-up, DMA loads the coefficients from internal memory into coefficient memory.

Internal Memory Storage

This section describes the required storage model for the IIR accelerator.

Coefficient Memory Storage

Coefficients and Dk values for a particular biquad BQD[k] should be stored in internal memory in the order Ak0, Ak1, Bk1, Ak2, Bk2, Dk2, Dk1.



The naming convention for the filter coefficients used here is different from the one used in MATLAB. The following conversion should be used when using MATLAB generated coefficients:
(Akx = bx and Bkx = -ax).

In other words, the coefficients for each biquad should be stored in the order:

b0, b1, -a1, b2, -a2, dk2, dk1

For N biquad stages, the order of coefficients should be as follows:

```
b01, b11, -a11, b21, -a21, dk21, dk11,  
b02, b12, -a12, b22, -a22, dk22, dk12  
.....  
b0N, b1N, -a1N, b2N, -a2N, dk2N, dk1N.
```

where b_{xN} and a_{xN} are the coefficients ($[b, a]$) for the N th biquad stage.

Operating Modes

The accelerator can be operated in the following modes.

Window Processing

Sample based processing mode is selected by configuring window size to 1. In this mode, one sample from a particular channel is processed through all the biquads of that channel and the final output sample is calculated.

In window based mode, multiple output samples (up to 1024) equal to the window size of that channel are calculated. After these calculations are complete, the accelerator begins processing the next channel. A configurable window size parameter is provided to specify the length of the window.

40-Bit Floating-Point Mode

In 40-bit floating-point mode, the input data/coefficient is treated as a 40-bit floating-point number. 40-bit floating-point MAC operations generate 40-bit results. This mode can be selected by setting bit 12 of the IIRCTL1 register.

Since the DMA bus width is only 32 bits, in 40-bit mode the IIR accelerator performs two packed 32-bit accesses to the memory to fetch one 40-bit input or coefficient data, or to store one 40-bit output word. The

IIR Accelerator

first 32-bit word provides the lower 32 bits and the 8 LSBs of the second 32-bit word provides rest of the upper 8 bits of the a complete 40-bit word. [Figure 6-12](#) shows the 32-40 bit packing used by accelerator.

-  Overheads might be required to pack the input 40-bit data into the format acceptable by the IIR accelerator and for unpacking the output of accelerator to the format acceptable by the rest of the application.

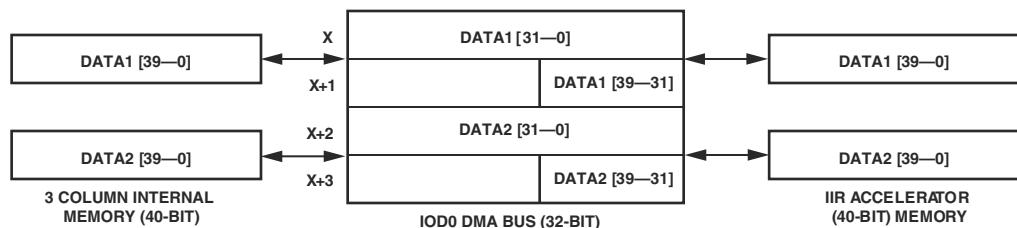


Figure 6-12. 32- to 40-Bit Packing

Data Transfers

The IIR filter works exclusively through DMA.

DMA Access

The IIR accelerator has two DMA channels (accelerator input and output) to connect to the internal memory. The DMA controller fetches the data and coefficients from memory and stores the result.

Chain Pointer DMA

The DMA controller supports circular buffer chain pointer DMA. One transfer control block (TCB) needs to be configured for each channel. The TCB contains:

- A control register value to configure the filter parameters for each channel
- DMA parameter register values for the input data (delay line)
- DMA parameter register values for coefficient load
- DMA parameter register values for output data

As shown in “[IIR Accelerator TCB](#)” on page 2-17 and [Figure 6-7](#), the accelerator loads the TCB into its internal registers and uses these values to fetch coefficients and data and to store results. After processing a window of data for any channel, the accelerator writes back the `IIRII` (input index register) and `IIROI` (output index register) values to the TCB in memory, so that data processing can begin from where it left off during the next time slot of that channel.

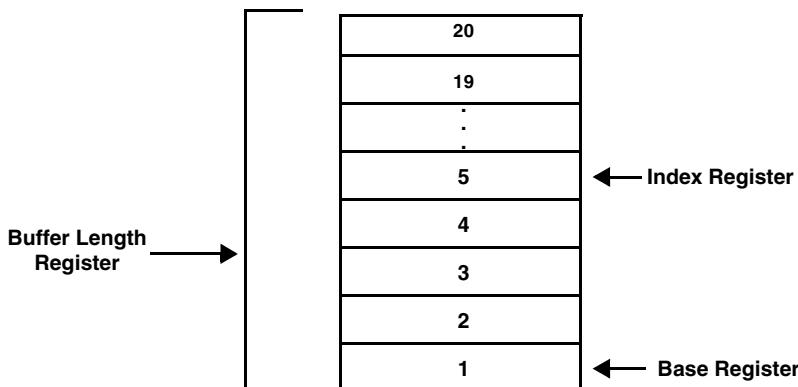


Figure 6-13. Circular Buffer Addressing

For 32-bit mode, the write back values for the index registers is equal to `IIRII + W` and `IIROI + W`.

For 40-bit mode, the write back values are:
 $\text{IIRII} + 2 \times W$ and $\text{IIROI} + 2 \times W$.

Accelerator input and output channels are used to connect to internal memory.



Note that the `IIRCTL2` register is part of the IIR TCB. This allows to program individual IIR channels having different control attributes.

Interrupts

The IIR accelerator has two interrupts that are programmable through the `PICR` registers ([Appendix B, Peripheral Interrupt Control](#)). The `ACCOI` and `ACC1I` source bits are used to connect IIR interrupts to the peripheral interrupt inputs of the core.

Table 6-4. Overview of IIR Interrupts

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
IIR (2 channels)	- Window Complete - Channel Complete - MAC status	- Internal transfer completion	RTI instruction	Need to route ACCxI (PICRx) to any PxxI

One interrupt line is shared by all the DMA interrupts and the other by MAC status interrupts. Separate status registers are provided to further differentiate the various sources.

Interrupt Sources

There are two interrupt sources associated with the accelerator. The `IIR_CCINTR` bit in the `IIRCTL1` register controls these interrupts. When set, the bit generates window complete interrupt and when cleared (default), an interrupt is generated after all the channels are complete.

Window Complete Interrupt – This interrupt is generated at the end of each channel when all the output samples are calculated corresponding to a window and updated index values are written back.

Channels Complete Interrupt – This interrupt is generated when all the channels are complete or when one iteration of time slots completes.

MAC Status Interrupt – The status interrupt sources are derived from the `IIRMACSTAT` register. For more information, see “[IIR MAC Status Register \(IIRMACSTAT\)](#)” on page A-91.

Service Channel Interrupts – Based on the `IIR_CCINTR` bit in the `IIRCTL1` register, both bits (`IIR_DMAWCDONE` or `IIR_DMAACDONE`) are set in the `IIRDMSTAT` register if either of the conditions is met. The interrupt service routine should read (to clear) both bits.

Service MAC Status Interrupts – A MAC status interrupt is generated whenever a floating-point operation results in an arithmetic exception. Reading the `IIRMACSTAT` register returns for which MAC unit is causing an exception.

Debug Features

The following sections describe the debugging features available on the accelerator.

Local Memory Access

The contents of IIR delay line and coefficient memories are made observable for debug by setting the `IIR_DBGMODE`/`IIR_DBGMEM` and `IIR_HLD` bits in the `IIRDEBUGCTL` control register. The debug address register (`IIRDBGADDR`) and four data registers are provided for debug operations. Bit 11 of the `IIRDBGADDR` register selects coefficient memory if set (=1) and selects delay line memory in cleared (=0).

The 40-bit wide debug mode read data register is organized as:

- The `IIRDBGRDDATA_L` register holds the lower 32 bits
- The `IIRDBGRDDATA_H` register holds the upper 8 bits

The 40-bit wide debug mode write data register is organized as:

- The `IIRDBGWRDATA_L` register holds the lower 32 bits and
- The `IIRDBGWRDATA_H` register holds the upper 8 bits

A read from the `IIRDBGRRDATA_L` register followed by a read from the `IIRDBGRRDATA_H` register returns the content of the 40-bit memory location pointed to by the address register. Data can be written into any memory location using the `IIRDBGWRDATA_L` register followed by the `IIRDBGWRDATA_H` register.

If the address auto increment bit (`IIR_ADRINC`) is set, the address register auto increments on `IIRDBGWRDATA_H/L` writes and `IIRDBGRRDATA_H/L` reads. During auto increment, the `IIR_DBGADDR` register cannot cross the data memory/coefficient memory boundary. The address boundary for data memory is 1024 locations and for coefficient memory 2048 locations

Single Step Mode

Programs can single step through the MAC operations and observe the memory contents after each step. The `IIR_DBGMODE/IIR_HLD` and `IIR_RUN` bits control the IIR MAC units.

Emulation Considerations

In IIR debug mode, the DMA operations are not observable.

Effect Latency

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

Write Effect Latency

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

IIR Accelerator Effect Latency

After the IIR registers are configured the effect latency is 1.5 PCLK cycles minimum and 2 PCLK cycles maximum. Writes to the PMCTL1 register have an effect latency of two PCLK cycles. Wait for at least four CCLK cycles after selecting another accelerator before accessing any of its registers.

IIR Throughput

Data throughput is one 32-bit data word per peripheral clock cycle for writes to memory, provided there are no conflicts. Read throughput from memory, throughput is one 32-bit data word per two peripheral clock cycles.

IIR throughput is calculated as follows:

Total number of peripheral clock cycles = $(TCB\ load + 5 \times B \times W) \times C$
where:

- B = number of bi-quads
- W = Window size
- C = number of channels
- TCB load = 36 PCLK cycles
- $5 \times B$ – Number of cycles to calculate B biquads (Note: This does not include the coefficient loading cycles as coefficients need to be loaded only once.)

Programming Model

The IIR supports up to 24 channels which are time division multiplexed (TDM). Each channel can have a maximum of 12 cascaded biquads. The window size for each channel is configurable using control registers. A window size of 1 corresponds to sample based operation and the maximum window size is 64.

The coefficients are initially stored in internal memory and one TCB per channel is created in internal memory with each channels' TCB pointing to the next channels'. The TCB also contains channel specific control registers, input data buffer parameters and output data buffer parameters.

-  The TCB of the last channel should point to the TCB of first channel.

The total number of channels is configured using the `IIRCTL1` register and DMA is enabled.

The procedure that the accelerator uses to process biquads is shown in [Figure 6-14](#) and described in the following procedure.

1. The controller loads all coefficients of all the channels into local storage.
2. Once all the coefficients are loaded, the controller goes to the first biquad of the first channel and calculates the output of the first biquad and updates the intermediate results for that biquad.
3. Then, the accelerator moves to the next biquad of that channel and repeats the process until all the biquads for that channel are completed and the results are stored to memory.
4. This process is repeated with next sample until one window of the corresponding channel is processed.
5. After one window of the channel accelerator is processed, the accelerator moves to the next channel and computes the results.

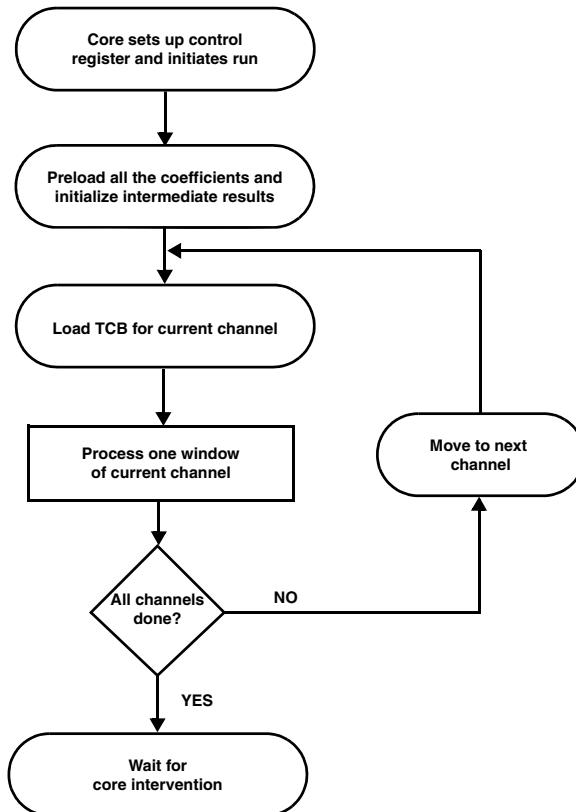


Figure 6-14. Biquad Processing Program Flow

Writing to Local Memory

1. Enable IIR module in PMCTL1 register.
2. Wait at least 4 CCLK cycles.
3. Clear the IIR_DMAEN bit in the IIRCTL1 register.
4. Set the IIR_DBGMODE, IIR_DBGMEM and IIR_HLD bits in the IIRDBGCTL register.

5. Set the `IIR_DRINC` bit in `IIRDEBUGCTL` register for address auto increment.
6. Write start address to the `IIRDBGADDR` register. If bit 11 is set, coefficient memory is selected.
7. Wait at least 4 CCLK cycles.
8. Write data to the `IIRDBGWRDATA_L` register.
9. Write data to the `IIRDBGWRDATA_H` register.

Reading from Local Memory

1. Enable IIR module in `PMCTL1` register.
2. Wait at least 4 CCLK cycles.
3. Clear the `IIR_DMAEN` bit in the `IIRCTL1` register.
4. Set the `IIR_DBGMODE`, `IIR_DBGMEM` and `IIR_HLD` bits in the `IIRDEBUGCTL` register.
5. Set the `IIR_DRINC` bit in the `IIRDEBUGCTL` register for address auto increment.
6. Write start address to the `IIRDBGADDR` register. If bit 11 is set, coefficient memory is selected.
7. Wait at least 4 CCLK cycles.
8. Read data from the `IIRDBGRDDATA_L` register.
9. Read data from the `IIRDBGRDDATA_H` register.

Single Step Mode

Single step mode can be used for debug purposes. An additional debug register is used in this mode.

1. Enable stop DMA during breakpoint hit in the emulator settings.
2. Clear the `IIR_HLD` bit and enable `IIR_DBGMODE` and `IIR_RUN` bits in `IIRDEBUGCTL` register.
3. Program FIR module according to the application.
4. In single step each iteration is updated in the emulator session.

Programming Example

In this example, an application needs IIR filtering for two channels of data; channel 1 has six biquads and channel 2 has eight biquads. The window size for all channels is 32.

1. Create a circular buffer in internal memory for each channel's data. The buffer should be large enough to avoid overwriting data before it is processed by the accelerator.
2. Configure internal memory buffers containing the 6×5 coefficients and the 6×2 Dk values for the channel 1 biquads, and the 8×5 coefficients and 8×2 Dk values of the channel 2 biquads.
3. Configure two TCBs in internal memory with each channel's chain pointer entry pointing to the next channel's and the last channel's chain pointer entry pointing to the first in a circular fashion.
4. Program the `IIRCTL2` register to use channel 1 TCB for 6 biquads and a window size of 32, and channel 2 for 8 biquads and a window size of 32.

5. Configure the index, modifier, and length entries in the TCBs to point to the corresponding channel's data buffer, coefficient buffer and output data buffer.

The location of the first channel's TCB is written to the chain pointer register in the accelerator.

6. Program the global control register `FIR_NCH` bit for 2 channels.
 - a. The accelerator starts and loads the first channel's TCB, loads coefficients and Dk values of all the 6 biquads into local storage, then loads the TCB of the second channel, and finally loads coefficients and Dk values of all the 8 biquads.
 - b. Once all the coefficients and Dk values are loaded, the controller loads the TCB of first channel and fetches the input sample. It then starts calculating the first biquad of the first channel.
 - c. The accelerator calculates the output of the first biquad and then updates the intermediate results for that biquad. Then it moves to the next biquad of that channel and repeats the biquad processing until all the biquads for that channel are done and the final result is stored to memory.
 - d. The accelerator repeats this process with next sample until one window of the corresponding channel is processed. Once the window is done, the accelerator saves the index values to memory and moves to the next channel. After both channels are done, the accelerator waits for core intervention.

7 PULSE WIDTH MODULATION

Pulse width modulation (PWM) is a technique for controlling analog circuits with a microprocessor's digital outputs. PWM is employed in a wide variety of applications, ranging from measurement to communications to power control and conversion. The interface specifications are shown in [Table 7-1](#).

Table 7-1. PWM Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	Yes, (External port)
SRU DAI Required	No
SRU DAI Default Routing	N/A
Interrupt Default Routing	No
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	N/A
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A
Access Type	
Data Buffer	No
Core Data Access	N/A

Features

Table 7-1. PWM Specifications (Cont'd)

Feature	Availability
DMA Data Access	N/A
DMA Channels	N/A
DMA Chaining	N/A
Boot Capable	N/A
Local Memory	No
Clock Operation	f_{PCLK}

Features

The following is a brief summary of the features of this interface.

- Four independent PWM units
- 2-phase output timing unit
- Center or edge aligned PWM
- Single or double update PWM timer period
- Output logic allows redirection of 2-phase output timing
- PWM units can operate synchronized to each other
- Complementary outputs allows bridge based applications

A block diagram of the module is shown in [Figure 7-1](#). The generation of the four output PWM signals on pins AH to BL is controlled by four primary blocks.

- The two-phase PWM timing unit, which is the core of the PWM controller, generates two pairs of complemented center based PWM signals.

- The emergency dead time insertion is implemented after the ‘ideal’ PWM output pair, including crossover, is generated.
- The output control unit allows the redirection of the outputs of the two-phase timing unit for each channel to either the high-side or the low-side output. In addition, the output control unit allows individual enabling/disabling of each of the four PWM output signals.
- The PWM interrupt controller generates an interrupt at the start of the PWM period which is shared for all modules.

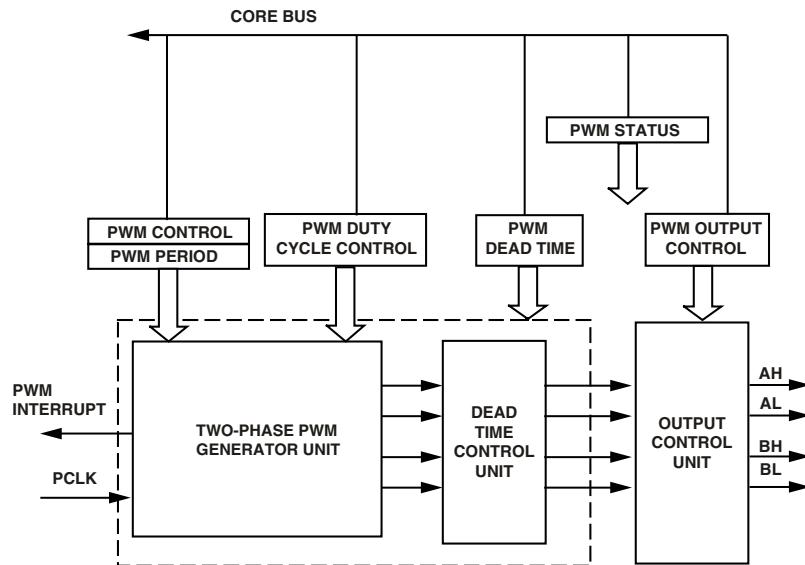


Figure 7-1. PWM Module Block Diagram

Pin Descriptions

The PWM module has four groups of four PWM outputs each, for a total of 16 PWM outputs. These outputs are described in [Table 7-2](#).

Table 7-2. PWM Pin Descriptions

Multiplexed Pin Name	Direction	Description
PWM_AH3-0	O	PWM output of pair A produce high side drive signals.
PWM_AL3-0	O	PWM output of pair A produce low side drive signals. Note in paired mode, this pin is the complement of AH3-0.
PWM_BH3-0	O	PWM output of pair B produce high side drive signals.
PWM_BL3-0	O	PWM output of pair A produce low side drive signals. Note in paired mode, this pin is the complement of BH3-0.

Multiplexing Scheme

By default the PWM output pins are disabled. To enable the PWM units refer to [Table 23-15 on page 23-30](#). [Table 7-3](#) and [Table 7-3](#) show the connection to the PWM outputs on the external port pins. [For more information, see “Pin Multiplexing” on page 23-28](#).

Table 7-3. PWM Connections

PWM Unit	Pin Multiplexing
PWM0	AMI_ADDR8 = AL0 AMI_ADDR9 = AH0 AMI_ADDR10 = BL0 AMI_ADDR11 = BH0
PWM1	AMI_ADDR12 = AL1 AMI_ADDR13 = AH1 AMI_ADDR14 = BL1 AMI_ADDR15 = BH1

Table 7-3. PWM Connections (Cont'd)

PWM Unit	Pin Multiplexing
PWM2	AMI_ADDR16 = AL2 AMI_ADDR17 = AH2 AMI_ADDR18 = BL2 AMI_ADDR19 = BH2
PWM3	AMI_ADDR20 = AL3 AMI_ADDR21 = AH3 AMI_ADDR22 = BL3 AMI_ADDR23 = BH3

SRU Programming

The ADSP-2147x and 2148x can output the PWM units 3–1 over the DPI pins. The `PWMONDPIEN` bit (bit 30 in the `SYSCTL` register) enables the routing output logic for the DPI group B register.

Register Overview

This section provides brief descriptions of the major registers. For complete register information, see [Appendix A, Registers Reference](#).

- **PWM global control register (PWMGCTL).** Enables or disables the four PWM groups simultaneously in any combination for synchronization between the PWM groups.
- **PWM global status register (PWMGSTAT).** Provides the status of each PWM group.
- **PWM control registers (PWMCTLx).** Used to set the operating modes of each PWM block. This register also allows programs to disable interrupts from individual groups.

Clocking

- PWM status registers (PWMSTATx). Report the phase and mode status for each PWM group.



The traditional read-modify-write operation to enable/disable a peripheral is different for the PWMs. [For more information, see “Global Control Register \(PWMGCTL\)” on page A-67.](#)

Clocking

The fundamental timing clock of the PWM controllers is peripheral clock (PCLK).

Functional Description

The individual elements shown in [Figure 7-1](#) are described in detail in the following sections.

Two-Phase PWM Generator

Each PWM group is able to generate complementary signals on two outputs in paired mode or each group can provide independent outputs in non-paired mode.

Switching Frequencies

The 16-bit read/write PWM period registers, PWMPERIOD3-0, control the PWM switching frequency.



The PWM generator does not support external synchronization mode.

The fundamental timing unit of the PWM controller is PCLK. Therefore, for a 200 MHz peripheral clock, the fundamental time increment is 5 ns. The value written to the PWMPERIODx register is effectively the number of

`PCLK` clock increments in a PWM period (edge aligned mode) or in a half PWM period (center aligned mode) in half a PWM period.

Therefore, the PWM switching period, T_s , can be written as:

$$T_s = 2 \times \text{PWMTM} \times t_{\text{PCLK}} \text{ (edge aligned)}$$

$$T_s = \text{PWMTM} \times t_{\text{PCLK}} \text{ (center aligned)}$$

For example, for a 200 MHz `PCLK` and a desired PWM center aligned switching frequency of 10 kHz ($T_s = 100 \mu\text{s}$), the correct value to load into the `PWMPeriodx` register is:

$$\text{PWMPeriod} = \frac{200 \times 10^6}{2 \times 10 \times 10^3} = 10000$$

The largest value that can be written to the 16-bit `PWMPeriodx` register is 0xFFFF = 65,535 which corresponds to a minimum PWM switching frequency of:

$$f_{(PWM),min} = \frac{200 \times 10^6}{2 \times 65535} = 1523 \text{ Hz}$$



`PWMPeriod` values of 0 and 1 are not defined and should not be used when the PWM outputs or PWM sync is enabled.

Duty Cycles

The two 16-bit read/write duty cycle registers, `PWMA` and `PWMB`, control the duty cycles of the four PWM output signals on the PWM pins. The two's-complement integer value in the `PWMA` register controls the duty cycle of the signals on the `PWM_AH` and `PWM_AL`. The two's-complement integer value in the `PWMB` register controls the duty cycle of the signals on `PWM_BH` and `PWM_BL` pins. The duty cycle registers are programmed in two's-complement integer counts of the fundamental time unit, `PCLK`, and define the desired on-time of the high-side PWM signal produced by the

Functional Description

two-phase timing unit over half the PWM period. The duty cycle register range is from:

$$(-\text{PWPERIOD} \div 2 - \text{PWMDT}) \text{ to } (+\text{PWPERIOD} \div 2 + \text{PWMDT})$$

which, by definition, is scaled such that a value of 0 represents a 50% PWM duty cycle. The switching signals produced by the two-phase timing unit are also adjusted to incorporate the programmed dead time value in the `PWMDT` register. The two-phase timing unit produces active low signals so that a low level corresponds to a command to turn on the associated power device.

A typical pair of PWM outputs (in this case for `PWM_AH` and `PWM_AL`) from the timing unit are shown in [Figure 7-2](#) for operation in single update mode. All illustrated time values indicate the integer value in the associated register and can be converted to time by simply multiplying by the fundamental time increment, (`PCLK`) and comparing this to the two's-complement counter. Note that the switching patterns are perfectly symmetrical about the midpoint of the switching period in single update mode since the same values of the `PWMMAX`, `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 7-2](#), 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 `PWMMAX` registers. Both switching edges are moved by an equal amount (`PWMDT` \times `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.

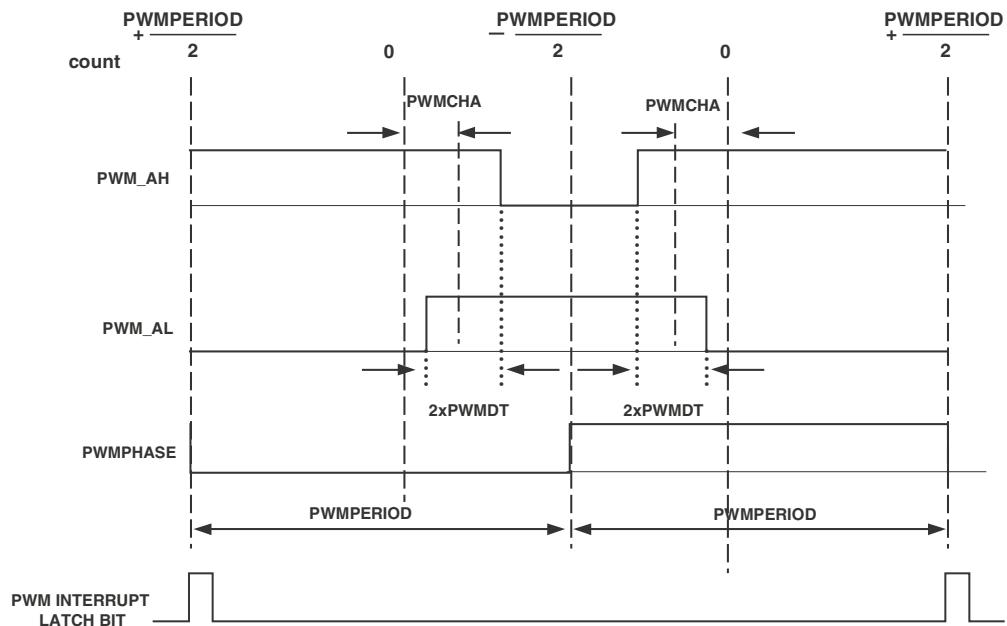


Figure 7-2. Center-Aligned Paired PWM in Single Update Mode, Low Polarity

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 7-2](#) may be written as:

The range of T_{AH} is:

$$[0 - 2 \times PWMPERIOD \times t_{PCLK}]$$

and the corresponding duty cycles are:

$$T_{AH} = (PWMPERIOD - 2 \times (PWMCHA + PWMDT) \times t_{PCLK})$$

Functional Description

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 7-3](#). 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 7-3](#) shows that the dead time is inserted into the PWM signals in the same way as in single update mode.

In general, the on-times (active low) of the PWM signals over the full PWM period in double update mode can be defined as:

$$T_S = (PWMPERIOD_1 + PWMPERIOD_2) \times t_{PCLK}$$

$$T_{AL} = \left(\frac{PWMPERIOD_1}{2} + \frac{PWMPERIOD_2}{2} - PWMCHA_1 - PWMCHA_2 - PWMDT_1 - PWMDT_2 \right) \times t_{PCLK}$$

$$T_{AH} = \left(\frac{PWMPERIOD_1}{2} + \frac{PWMPERIOD_2}{2} + PWMCHA_1 + PWMCHA_2 - PWMDT_1 - PWMDT_2 \right) \times t_{PCLK}$$

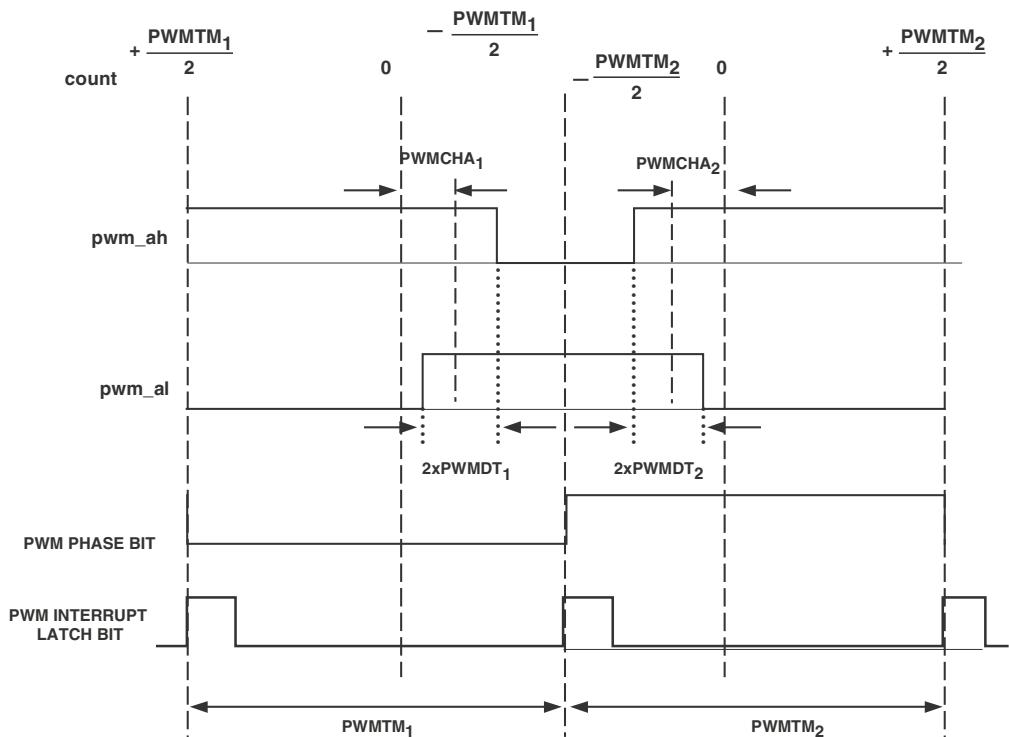


Figure 7-3. Center-Aligned Paired PWM in Double Update Mode, Low Polarity

where subscript 1 refers to the value of that register during the first half cycle and subscript 2 refers to the value during the second half cycle. The corresponding duty cycles are:

$$d_{AL} = \frac{T_{AL}}{T_S} = \frac{1}{2} - \frac{(PWMCHA_1 + PWMCHA_2 + PWMDT_1 + PWMDT_2)}{(PWMPERIOD_1 + PWMPERIOD_2)}$$

Functional Description

$$d_{AH} = \frac{T_{AH}}{T_H} = \frac{1}{2} + \frac{(PWMC_{HA_1} + PWMC_{HA_2} - PWMDT_1 - PWMDT_2)}{(PWMPERIOD_1 + PWMPERIOD_2)}$$

since for the general case in double- update mode, the switching period is given by:

$$T_S = (PWMPERIOD_1 + PWMPERIOD_2) \times t_{PCLK}$$

Again, the values of T_{AH} and T_{AL} are constrained to lie between zero and T_S . Similar PWM signals to those illustrated in [Figure 7-2](#) and [Figure 7-3](#) can be produced on the BH and BL outputs by programming the $PWMBx$ registers in a manner identical to that described for the $PWMX$ registers.

Dead Time

The second important parameter that must be set up in the initial configuration of the PWM block is the switching dead time. This is a short delay time introduced between turning off one PWM signal (say AH) and turning on the complementary signal, AL . This short time delay is introduced to permit the power switch being turned off (AH in this case) to completely recover its blocking capability before the complementary switch is turned on. This time delay prevents a potentially destructive short-circuit condition from developing across the DC link capacitor of a typical voltage source inverter.

The 10-bit, read/write $PWMDT3-0$ registers control the dead time. The dead time, T_d , is related to the value in the $PWMDTx$ registers by:

$$T_d = PWMDT \times 2 \times t_{PCLK}$$

Therefore, a $PWMDT$ value of 0x00A (= 10), introduces a 200 ns delay between when the PWM signal (for example AH) is turned off and its complementary signal (AL) is turned on. The amount of the dead time can

therefore be programmed in increments of $2 \times \text{PCLK}$ (or 10 ns for a 200 MHz peripheral clock). The `PWMDTx` registers are 10-bit registers, and the maximum value they can contain is 0x3FF (= 1023) which corresponds to a maximum programmed dead time of:

$$T_{d,max} = 1023 \times 2 \times t_{\text{PCLK}} = 1023 \times 2 \times 10 \times 10^{-9} = 10.2\mu\text{s}$$

This equates to an `PCLK` rate of 200 MHz. Note that dead time can be programmed to zero by writing 0 to the `PWMDTx` registers (see “[Pulse Width Modulation Registers](#)” on page [A-67](#)).

Output Control Unit

The `PWMSEG` register contains four bits (0 to 3) that can be used to individually enable or disable each of the 4 PWM outputs.

Output Enable

If the associated bit of the `PWMSEG` register is set (=1), then the corresponding PWM output is disabled, regardless of the value of the corresponding duty cycle register. This PWM output signal remains disabled as long as the corresponding enable/disable bit of the `PWMSEGx` register is set. In single update mode, changes to this register only become effective at the start of each PWM cycle. In double update mode, the `PWMSEG` register can also be updated at the mid-point of the PWM cycle.



After reset, all four enable bits of the `PWMSEG` register are cleared so that all PWM outputs are enabled by default.

Output Polarity

The polarity of the generated PWM signals is programmed using the `PWMPOLARITY3-0` registers, so that either active high or active low PWM patterns can be produced. The polarity values can be changed on the fly if

Functional Description

required, provided the change is done a few cycles before the next period change.

Complementary Outputs

The PWM controller can be operated in paired or non paired mode (PWMCTLx register).

In non paired mode (default) both outputs (high and low side) are driven independently. Since paired mode drives the output logic of the PWM in a complementary fashion (low side = /high side), this feature may be useful in PWM bridge applications.

Crossover

The PWMSEG3-0 registers contain two bits (AHAL_XOVR and BHBL_XOVR), one for each PWM output. If crossover mode is enabled for any pair of PWM signals, the high-side PWM signal from the timing unit (for example, AH) is diverted to the associated low side output of the output control unit so that the signal ultimately appears at the AL pin.

The corresponding low side output of the timing unit is also diverted to the complementary high side output of the output control unit so that the signal appears at the AH pin. Following a reset, the two crossover bits are cleared so that the crossover mode is disabled on both pairs of PWM signals. Even though crossover is considered an output control feature, dead time insertion occurs after crossover transitions to eliminate shoot-through safety issues.

Note that crossover mode does not work if:

1. One signal of PWM_AL-PWM_AH or PWM_BL-PWM_BH is disabled.
2. PWM_AL and PWM_AH or PWM_BL and PWM_BH have different polarity settings from PWMPOLx registers.

In other words, both `PWM_AL` and `PWM_AH` or `PWM_BL` and `PWM_BH` should be enabled and both should have same polarity for proper operation of cross-over mode.

Emergency Dead Time for Over Modulation

The PWM timing unit is capable of producing PWM signals with variable duty cycle values at the PWM output pins. At the extreme side of the modulation process, settings of 0% and 100% modulation are possible. These two modes are termed full OFF and full ON respectively.



Full OFF and full ON over-modulation is entered by virtue of the commanded duty cycle values in conjunction with the setting in the `PWMDTX` registers. Settings that fall between the extremes are considered normal modulation. These settings are explained in more detail below.

Full On. The PWM for any pair of PWM signals operates in full on when the desired high side output of the two-phase timing unit is in the on state (low) between successive PWM interrupts.

Full Off. The PWM for any pair of PWM signals operates in full off when the desired high side output of the two-phase timing unit is in the off state (high) between successive `PWMSYNC` pulses.

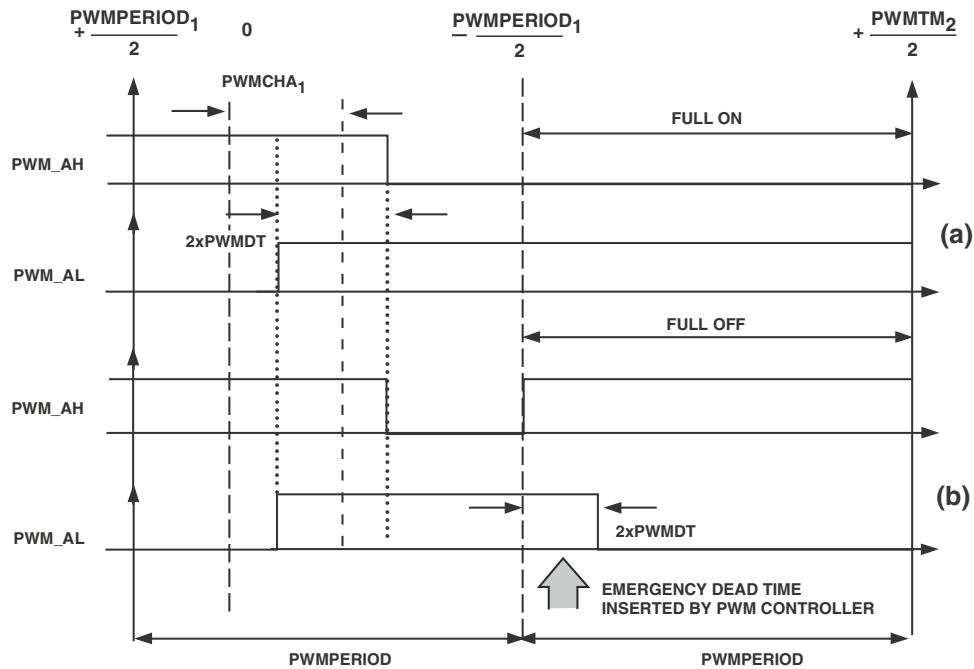
Normal Modulation. The PWM for any pair of PWM signals operates in normal modulation when the desired output duty cycle is other than 0% or 100% between successive `PWMSYNC` pulses.

There are certain situations, when transitioning either into or out of either full on or full off, where it is necessary to insert additional *emergency dead time* delays to prevent potential *shoot-through conditions* in the inverter. These transitions are detected automatically and, if appropriate, the emergency dead time is inserted to prevent the shoot through conditions.

Functional Description

Inserting additional emergency dead time into one of the PWM signals of a given pair during these transitions is only needed if both PWM signals would otherwise be required to toggle within a dead time of each other. The additional emergency dead time delay is inserted into the PWM signal that is toggling into the on state. In effect, the turn on (if turning on during this dead time region), of this signal is delayed by an amount of $2 \times \text{PWMDT} \times \text{PCLK}$ from the rising edge of the opposite output. After this delay, the PWM signal is allowed to turn on, provided the desired output is still scheduled to be in the on state after the emergency dead time delay.

[Figure 7-4](#) illustrates two examples of such transitions. In (a), when transitioning from normal modulation to full on at the half cycle boundary in double update mode, no special action is needed. However in (b), when transitioning into full off at the same boundary, an additional emergency dead time is necessary. This inserted dead time is a little different 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.



(a) TRANSITION FROM NORMAL MODULATION TO FULL-ON, AT HALF-CYCLE BOUNDARY IN DOUBLE UPDATE MODE, WHERE NO ADDITIONAL DEAD TIME IS NEEDED.

(b) TRANSITION FROM NORMAL MODULATION TO FULL-OFF, AT HALF-CYCLE BOUNDARY IN DOUBLE UPDATE MODE, WHERE ADDITIONAL DEAD TIME IS INSERTED BY THE PWM CONTROLLER

Figure 7-4. Normal Modulation to Full ON to Full OFF Transition

Output Control Feature Precedence

The order in which output control features are applied to the PWM signal is significant and important. The following lists the order in which the signal features are applied to the PWM output signal.

1. Duty Cycle Generation
2. Crossover
3. Output Enable

Operation Modes

4. Emergency Dead Time Insertion
5. Output Polarity

Operation Modes

The following sections provide information on the operating modes of the PWM module.

Waveform Modes

The PWM module can operate in both edge- and center-aligned modes. These modes are described in the following sections.

Edge-Aligned Mode

In edge-aligned mode, shown in [Figure 7-5](#), the PWM waveform is left-justified in the period window. A duty value of zero, programmed through the `PWMMAX` registers, produces a PWM waveform with 50% duty cycle. For even values of period, the PWM pulse width is exactly $\text{period}/2$, whereas for odd values of period, it is equal to $\text{period}/2$ (rounded up). Therefore for a duty value programmed in two's-complement, the PWM pulse width is given by:

To generate constant logic high on PWM output, program the duty register with the value $\geq +\text{period}/2$.

To generate constant logic low on PWM output, program the duty register with the value $\geq -\text{period}/2$.

For example, using an odd period of $p = 2n + 1$, the counter within the PWM generator counts as $(-n\dots0\dots+n)$. If the period is even ($p = 2n$) then the counter counts as $(-n+1\dots0\dots n)$.

The PWM switching period time for edge aligned mode is:
 $T_s = t_{PCLK} \times \text{PWM_PERIOD}$.

For more information see “[Pulse Width Modulation Registers](#)” on page [A-67](#) .

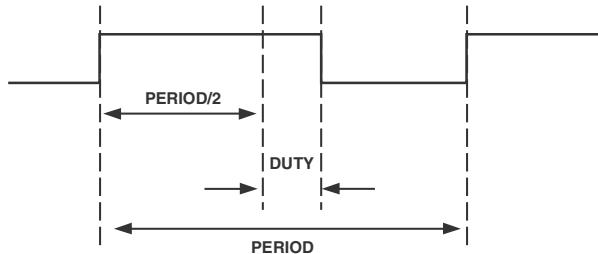


Figure 7-5. Edge Aligned PWM Wave with High Polarity

Center-Aligned Mode

Most of the following description applies to paired mode, but can also be applied to non-paired mode, the difference being that each of the four outputs from a PWM group is independent. Within center aligned mode, shown in [Figure 7-6 on page 7-21](#) there are several options to choose from.

Center-Aligned Single Update Mode. Duty cycle values are programmable only once per PWM period, so that the resultant PWM patterns are symmetrical about the mid-point of the PWM period.

Center-Aligned Double Update Mode. Duty cycle values are programmable only twice per PWM period. This second updating of the PWM registers is implemented at the mid-point of the PWM period, producing asymmetrical PWM patterns that produce lower harmonic distortion in two-phase PWM inverters.

Center-Aligned Paired Mode. Generates complementary signals on two outputs.

Operation Modes

Center-Aligned Non-Paired Mode. Generates independent signals on two outputs.

In paired mode, the two's-complement integer values in the 16-bit read/write duty cycle registers, `PWMAX` and `PWMBX`, control the duty cycles of the four PWM output signals on the `PWM_AL`, `PWM_AH`, `PWM_BL` and `PWM_BH` pins respectively. The duty cycle registers are programmed in two's-complement integer counts of the fundamental time unit, `PCLK` and define the desired on time of the high side PWM signal over one-half the PWM period.

The duty cycle register range is from $(-\text{PWMPERIOD}/2 - \text{PWMDT})$ to $(+\text{PWMPERIOD}/2 + \text{PWMDT})$, 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. The program first reads the global status register and clears all the intended bits by explicitly writing 1.

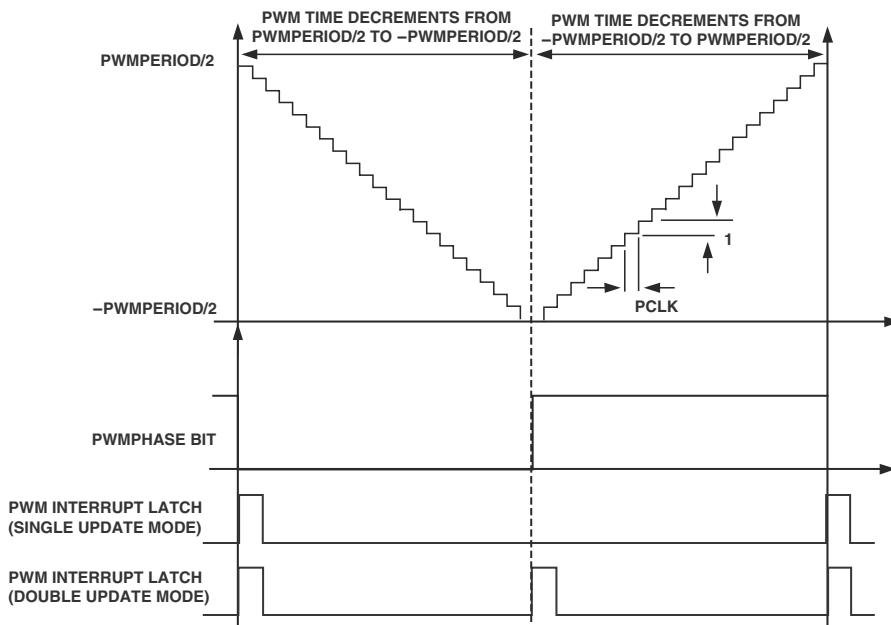


Figure 7-6. Operation of Internal PWM Timer (Center Aligned)

PWM Timer Edge Aligned Update

The internal operation of the PWM generation unit is controlled by the PWM timer which is clocked at the peripheral clock rate, PCLK . The operation of the PWM timer over one full PWM period is illustrated in [Figure 7-7](#). It can be seen that during the first half cycle, the PWM timer decrements from $\text{PWMPERIOD}/2$ to 0 using a two's-complement count.

At this point, the count direction changes and the timer continues to increment from 0 to the $\text{PWMPERIOD}/2$ value.

Also shown in [Figure 7-7](#) are the PWM interrupt pulses for operation in edge aligned mode. An PWM interrupt is latched at the beginning of every PWM cycle. Note that the PWMSTAT register has no meaning in this mode and is always set.

Operation Modes

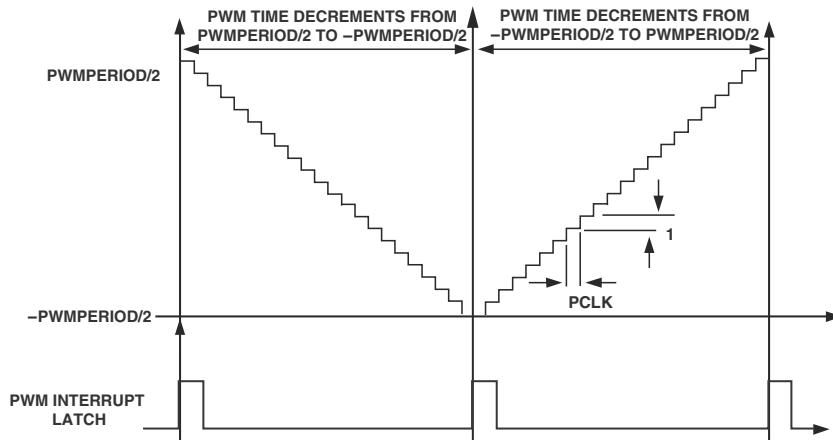


Figure 7-7. Operation of Internal PWM Timer (Edge Aligned)

Single Update Mode

In single update mode, a single PWM interrupt is produced in each PWM period. The rising edge of this signal marks the start of a new PWM cycle and is used to latch new values from the PWM configuration registers (`PWMTM` and `PWMDT`) and the PWM duty cycle registers (`PWMCHx`) into the two-phase timing unit. In addition, the `PWMSEG` register is also latched into the output control unit on the rising edge of the PWM interrupt latch pulse. In effect, this means that the characteristics and resultant duty cycles of the PWM signals can be updated only once per PWM period at the start of each cycle. The result is that PWM patterns that are symmetrical about the mid-point of the switching period are produced.

Double Update Mode

In double update mode, there is an additional PWM interrupt latch pulse produced at the mid-point of each PWM period. The rising edge of this new PWM pulse is again used to latch new values of the PWM

configuration registers, duty cycle registers and the PWMSEG register. As a result, it is possible to alter both the characteristics (switching frequency and dead time) as well as the output duty cycles at the mid-point of each PWM cycle. Consequently, it is possible to produce PWM switching patterns that are no longer symmetrical about the mid-point of the period (asymmetrical PWM patterns).

In double update mode, it may be necessary to know whether operation at any point in time is in either the first half or the second half of the PWM cycle. This information is provided by the PWMPHASE bit of the PWMSTAT register which is cleared during operation in the first half of each PWM period (between the rising edge of the original PWM interrupt latch pulse and the rising edge of the new PWM interrupt pulse introduced in double update mode). The PWMPHASE bit of the the PWMSTAT register is set during operation in the second half of each PWM period. This status bit allows programs to make a determination of the particular half-cycle during implementation of the PWM interrupt service routine, if required.

The advantage of the double update mode is that the PWM process can produce lower harmonic voltages and faster control bandwidths are possible. However, for a given PWM switching frequency, the interrupts occur at twice the rate as in double update mode. Since new duty cycle values must be computed in each PWM interrupt service routine, there is a larger computational burden on the processor in the double update mode. Alternatively, the same PWM update rate may be maintained at half the switching frequency to give lower switching losses.

Effective Accuracy

The PWM has 16-bit resolution but accuracy is dependent on the PWM period. In single update mode, the same values of PWMA and PWMB are used to define the on times in both half cycles of the PWM period. As a result, the effective accuracy of the PWM generation process is $2 \times \text{PCLK}$ (or 10 ns for a 200 MHz clock). Incrementing one of the duty cycle registers by one changes the resultant on time of the associated PWM signals by $2 \times \text{PCLK}$.

Operation Modes

in each half period (or $2 \times \text{PCLK}$ for the full period). In double update mode, improved accuracy is possible since different values of the duty cycles registers are used to define the on times in both the first and second halves of the PWM period. As a result, it is possible to adjust the on-time over the whole period in increments of PCLK . This corresponds to an effective PWM accuracy of PCLK in double update mode (or 10 ns for a 200 MHz clock). The achievable PWM switching frequency at a given PWM accuracy is tabulated in [Table 7-4](#). In [Table 7-4](#), $\text{PCLK} = 200 \text{ MHz}$.

Table 7-4. PWM Accuracy in Single- and Double Update Modes

Resolution (bits)	Single Update Mode PWM Frequency (kHz)	Double Update Mode PWM Frequency (kHz)
8	$200 \text{ MHz} \div 2 \times 2^8 = 390.63$	$200 \text{ MHz} \div 2^8 = 781.25$
9	195.3	390.6
10	97.7	195.3
11	48.8	97.7
12	24.4	48.8
13	12.2	24.4
14	6.1	12.2

Synchronization of PWM Groups

The `PWMGCTL` register enables or disables the four PWM groups in any combination. This provides synchronization across the four PWM groups.

The `PWM_SYNC_ENx` bits in this register can be used to start the counter without enabling the outputs through `PWM_EN`. So when `PWM_ENx` is asserted, the 4 PWM outputs are automatically synced to the initially programmed period. In most cases, all `SYNC` bits can be initialized to zero, enabling the `PWM_ENx` bits of the four PWM groups at the same time synchronizes the four groups.

The PWM sync enable feature allows programs to enable the `PWN_SYNC_ENx` bits to independently start the main counter without enabling the corresponding PWM module using the `PWM_ENx` bits. To synchronize different groups, enable the corresponding group's `PWM_ENx` bit at the same time. In order to stop the counter both the `PWM_DISx` and `PWM_SYNC_DISx` bits should be set in this register.

Interrupts

The following sections provide information on the PWM and interrupt generation. [Table 7-5](#) provides an overview of PWM interrupts.

Table 7-5. PWM Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
PWM (Edge/center aligned, single/double update, 4 channels)	Period start		W1C (Write 1-to-clear) PWMGSTAT + RTI instruction	Need to route PWMI (PICRx) to any PxxI

Typically the PWM interrupt is used to periodically execute an interrupt service routine (ISR) to update the two PWM channel duties according to a control algorithm based on expected system operation. The PWM interrupt can trigger the ADC to sample data for use during the ISR. During processor boot the PWM is initialized and program flow enters a wait loop. When a PWM interrupt occurs, the ADC samples data, the data is algorithmically interpreted, and new PWM channel duties are calculated and written to the PWM. More sophisticated implementations include different startup, runtime, and shutdown algorithms to determine PWM channel duties based on expected behavior and further features.

During initialization, the `PWMTM` register is written to define the PWM period and the `PWMCHx` registers are written to define the initial channel pulse widths. The PWM interrupt is assigned to one of the core's User

Interrupts

interrupts and is unmasked in the core. The `PWMSEG` and `PWMCHx` registers are also written, depending on the system configuration and modes. During the PWM interrupt driven control loop, only the `PWMCHx` duty values are typically updated. The `PWMSEG` register may also be updated for other system implementations requiring output crossover.

For interrupt execution, the specific `PWM_IRQEN` bit in the corresponding `PWMCTLx` register must be set including the `IMASK` or `LIRPTL` registers based on the programmable interrupt to be used.

Whenever a period starts, the PWM interrupt is generated. The interrupt latch bit is set 1 `PCLK` cycle after the PWM counter resumes. Since all four PWM units share the same interrupt vector, the interrupt service routine should read the `PWMGSTAT` register in order to determine the source of the interrupt. Next, the ISR needs to clear the status bits of the `PWMGSTAT` register by explicitly writing 1 into the status bit (W1C) as shown in [Listing 7-1](#).

[Listing 7-1.](#) Writing 1 Into the Status Bit

```
GPWM_ISR:  
ustat2=dm(PWMGSTAT);           /* read global status reg */  
bit tst ustat2 PWM_STAT2;      /* test PWM2 status */  
if tf jump PWM2_ISR;          /* jump to PWM2 routine */  
instruction;  
instruction;  
  
PWM2_ISR:  
r1=PWM_STAT2;  
dm(PWMGSTAT)=r1; /* W1C to clear PWM2 interrupt */  
r10=dm(PWMCTL2); /* dummy read for write latency */  
instruction;  
rti;
```

Debug Features

The module contains four debug status registers (PWMDBG3-0), which can be used for debug aid. Each register is available per unit. The registers return current status information about the AH, AL, BH, BL output pins.

Status Debug Register

The module contains four debug status registers (PWMDBG3-0), which can be used for debug aid. Each register is available per unit. The registers return current status information about the AH, AL, BH, BL output pins.

Emulation Considerations

An emulation halt does not stop the PWM period counter.

Effect Latency

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

Write Effect Latency

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

PWM Effect Latency

After the PWM registers are configured the effect latency is 1 PCLK cycle minimum and 2 PCLK cycles maximum.

Effect Latency

8 MEDIA LOCAL BUS

Media Local Bus (MediaLB®) is an on-PCB or inter-chip communication bus, which allows an application to access the MOST network data. Media Local Bus supports all the MOST network data transport methods including synchronous stream data, asynchronous packet data, control message data and isochronous data. The media local bus topology supports communication among the MLB controller and MLB devices, where the MLB controller is the interface between the MLB devices and the MOST network.

More information on the MediaLB protocol can be found in the MediaLB Device Specification at www.smsc-ais.com.

Table 8-1 shows the interface specifications.

Table 8-1. MLB Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Default Routing	No

Table 8-1. MLB Specifications (Cont'd)

Feature	Availability
Protocol	
Master Capable	No
Slave Capable	Yes
Transmission Simplex	Yes
Transmission Half Duplex	Yes
Transmission Full Duplex	No
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	31
DMA Chaining	No
Interrupt Source	Core/DMA
Boot Capable	No
Local Memory	Yes
Clock Operation	50 MHz

The MLB module in the ADSP-214xx serves as an interface between the MediaLB and ADSP-214xx, implementing the requirements of the physical layer and the link layer outlined in the MediaLB specification. It supports up to 31 logical channels with up to 124 bytes of data per MediaLB frame. Transmit and receive data can be transferred between MediaLB and on-chip memory with single word core-driven transfers or with DMA block transfers.



The MLB interface on supports MOST25 and MOST50 data rates. Isochronous modes of transfer are not supported.

Features

The MLB device has the following features.

- Support for both 3-pin and 5-pin MediaLB interfaces
- Selectable MediaLB clock rate: 256Fs, 512Fs and 1024Fs
- Support for control, streaming and packet data
- Support for 31 channels, configured for any channel type (synchronous, asynchronous, control) and direction (transmit and receive)
- DMA and core-driven data transport methods
- Memory for channel data buffering
- System channel command handling
- Hardware loop-back test mode support
- Support for transmit command and data transmission
- Support for data reception and receive status response transmission
- Programmable threshold and depth for all buffers
- Support for Big-Endian and Little-Endian data formats
- Support for streaming channel frame synchronization

Pin Descriptions

The MediaLB pin descriptions can be found in the product specific data sheet.

Register Overview

The following sections provide brief descriptions of the registers used by the MediaLB interface. For complete bit descriptions, see “[Media Local Bus Registers](#)” on page A-94.

Device Configuration and Status Registers

These registers are used to set up the basic features of the interface and to report status of the MLB network. They include the following registers.

- **Device Control Configuration Register (MLB_DCCR).** Controls the device enable/disable, clock rate, lock status and addressing of the MLB.
- **System Status Register (MLB_SSCR), System Data Configuration Register (MLB_SDCR), System Mask Configuration Register (MLB_SMCR).** Controls system features of the MediaLB interface and system interrupt handling.
- **Synchronous Base Register (MLB_SBCR), Asynchronous Base Register (MLB_ABCR), Isynchronous Base Register (MLB_IBCR).** Defines the base address for control RX/TX system memory buffers.

Channel Configuration Registers

These registers are used to configure and monitor individual MLB channels. They include the following registers.

- **Channel Control Registers (MLB_CECRx).** Defines basic attributes about a given logical channel, such as the channel enable, channel type, channel direction, and channel address. The definition of the bit fields in this register vary depending on the selected channel type.

- **Channel Interrupt Status Register (MLB_CICR).** Reflects the channel interrupt status of the individual logical channels. These bits are set by hardware when a channel interrupt is generated. The channel interrupt bits are sticky and can only be reset by software.
- **Channel Status Configuration Registers (MLB_CSCRx).** Reflects the status of the current buffer and previous buffer for a given logical channel. The definition of the bit fields in this register vary dependant on the selected channel type.
- **Channel Current Buffer Configuration Registers (MLB_CCBCRx), Channel Next Buffer Configuration Registers (MLB_CNBCRx), Local Buffer Configuration Registers (MLB_LCBCRx).** These registers allow programs to control and monitor the buffers used in the MLB network.

Clocking

Media Local Bus Clock. This clock is generated by the MLB controller that is synchronized to the MOST network and provides the timing for the entire MLB interface at 49.152 MHz at $F_s=48\text{ kHz}$.

Functional Description

Once per MOST network frame, the MLB controller generates a unique frame sync pattern on the `MLBSIG` line. The end of the frame sync pattern defines the byte boundary and the channel boundary for the `MLBSIG` and `MLBDAT` lines of all MLB devices.

The MLB controller manages the arbitration for all the channels on the MLB and grants bandwidth for all the MLB devices. A MLB physical channel is defined as four bytes wide, or a quadlet. Physical channels can be grouped into multiple qualdets (which do not have to be consecutive)

Functional Description

to form a MLB logical channel, which is defined by a unique channel address.

As shown in [Figure 8-1](#), the MLB controller initiates communication by sending out a channel address on the MLBSIG line for each physical channel. The channel address indicates which MLB device is transmitting and which MLB devices are receiving in the following physical channel. Therefore, four bytes after the controller outputs the channel address on the MLBSIG line, the transmitting device outputs a command byte command on the MLBSIG line and outputs the respective data on the MLBDAT line, concurrently. The MLB command byte contains the type of data currently being transmitted (for example synchronous, asynchronous or control).

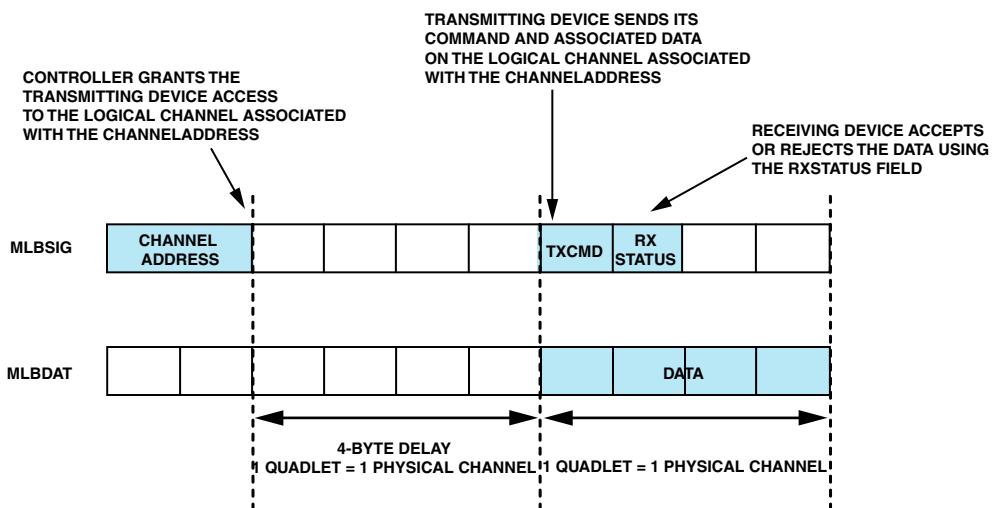


Figure 8-1. MLB Data Structure

The MLB device receiving the channel data outputs a status byte, RxStatus, on the MLBSIG line immediately after the transmitting device outputs the command byte. The status response can indicate that the receiving device is busy and cannot receive the data at present, or the device is ready to receive the data. Since synchronous stream data is sent in a broadcast

fashion, receiving devices cannot return a busy status and should not drive RxStatus onto the MLBSIG line.

For further information on the MediaLB Specification, please refer to the MediaLB specification document available on www.smsc-ais.com.

Operating Modes

The following sections describe the operating modes of the MLB interface.

Streaming Channel Frame Synchronization

Certain types of streaming applications require data to be synchronous with the MediaLB frame, including: stereo, 5.1 audio, and Generic Synchronous Packet Format (GSPF) DTCP. This feature is provided as an optional programmable synchronous logical channel by setting the FSE bit in the `MLB_CECRx` register. When enabled, the synchronous logical channel begins transmitting data only at a MediaLB frame boundary. A maskable interrupt is generated when the loss of frame synchronization occurs.

In order to use this option, system software must program the FSPC bits in the `MLB_CECRx` register with the expected number of physical channels per frame for the logical channel, and unmask the STS interrupt bit (bit 6) in the `MLB_CSCRx` register by setting the MASK bit in `MLB_CECRx` register. An interrupt is generated when the actual number of physical channels detected during a MLB frame does not match the expected value. Software can also set the FSCD bit in `MLB_CSCRx` register, which causes hardware to automatically disable a logical channel (clear the channel enable bit in the `MLB_CECRx` register) when synchronization is lost.

Frame synchronization is not supported for asynchronous, control or isochronous channels.

Data Transfer

Big-Endian and Little-Endian Mode

The byte order in which data is transferred between local channel buffers and internal memory is determined by enabling either big-endian or little-endian mode. Both data transfer methods, DMA and I/O mode support big-endian and little-endian system memory data formats. This option is selected using the `MLE` bit in the `MLB_DCCR` register.

Data Transfer

Two modes of operation are supported for transferring channel data between the MLB and internal memory. DMA allows the multi-channel DMA engine to manage data transfers without core intervention. Core driven mode (I/O mode) allows software to manage the transfer of data between MLB and internal memory.

All hardware channels must use the same data transfer method. Mixed mode operation where hardware channels operate in both I/O mode and DMA mode is not supported.

Core Driven Data Transfer

Core driven mode is an interrupt driven data transfer method between hardware channels and internal memory. When the MLB is configured in I/O mode, the `MLB_CCBCRx` and `MLB_CNBCRx` registers are used as the receive data buffer and transmit data buffer respectively. Transmit and receive service request interrupts are generated when data is to be transferred from/to internal memory to/from MLB local channel buffer.

I/O Local Channel Buffering

As shown in [Figure 8-2](#), once a quadlet is received on the MLB bus, it is transferred to the corresponding local channel buffer. The data transfer takes place from this local channel buffer to internal memory. The size of

the local channel buffer memory is 124 quadlets. At reset each channel has four quadlets each.

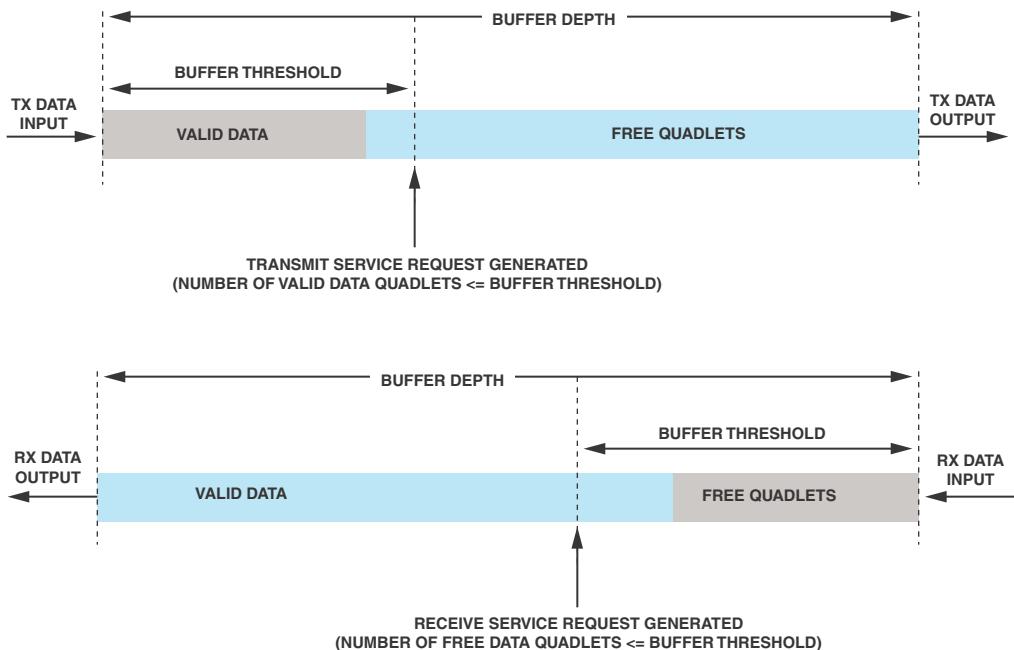


Figure 8-2. Local Channel Buffer Threshold Mechanism

The buffer depth, buffer threshold and buffer start address for each channel is programmed using the respective `MLB_LCBCRx` register. The buffer threshold is used by hardware channels in I/O mode only.

Hardware uses the local channel buffer threshold to determine when to issue an I/O service request to system software. I/O service requests are generated when:

- The number of valid quadlets in the local channel buffer falls below the threshold for transmit channels or:
 $\text{valid quadlets} \leq \text{MLB_LCBCRx}$ (programmed using `TH` bits)

Data Transfer

- The number of free quadlets in the local channel buffer falls below the threshold for receive channels or:
free quadlets \leq `MLB_LCBCRx` (programmed using `TH` bits)
- A receive channel detects a broken packet (ReceiverBreak, AsyncBreak, ControlBreak or ReceiverProtocolError).

Configuring local channel buffer memory is accomplished using the `MLB_LCBCRx` register. [For more information, see “Programming Model” on page 8-16.](#)

DMA

There are two modes of DMA—Ping-pong buffering and circular buffering.

There are 31 DMA channels for the 31 logical channels. Each channel can address up to 16k words.

The DMA address is comprised of:

- A 5-bit base configured in the MLB base register set (`MLB_SBCR`, `MLB_IBCR`, `MLB_ABCR` or `MLB_CBCR` for the corresponding channel data type). The 5-bit base is the same for all the channels of the same type (for example for all synchronous RX channels `MLB_SBCR31-16` act as the base).
- A 14-bit offset configured using the `BCA` bits in the `MLB_CCBCRx` register. The register holds the lower 14 bits (bits 31–18 for start address and 15–2 for end address). Bits 17–16 and bits 1–0 are reserved and must always be written with zero.

For example, if the internal address is 0xC0100, then 19 bits of the address translates to address 0x40100 because the internal memory offset for the ADSP-214xx is 0x0008 0000.

The lower 14 bits (00 0001 0000 0000) and the 2 reserved bits “00” are written in the `MLB_CNBCRx` register (write 0000 0100 0000 0000 = 0x0400 to bits 31–16 in the `MLB_CNBCRx` register for the start address or bits 15–0 for the end address). The remaining higher 5 bits (1 0000) are written in one of the the base address registers, depending on the transfer mode. For example, if using synchronous mode, write bits 31–16 = 0x0010 for receive and bits 15–0 for transmit in the `MLB_SBCR` register.

The base address registers and offset registers use round robin arbitration to determine which logical channel is granted access to the DMA bus. [For more information, see “Programming Model” on page 8-16.](#)

Ping-Pong DMA

Logical channels operate in ping-pong DMA mode when the channel mode select bits (`MLB_CECRx` register bits 25–26) = 00. When MLB is configured in ping-pong mode, the `MLB_CCBCRx` and `MLB_CNBCRx` registers are used to configure and monitor the system memory current buffer and next buffer respectively. Ping-pong DMA is available for all data types.

When receiving and transmitting asynchronous and control packet data, the current buffer and next buffer are independent internal memory buffers. This allows hardware to support the ping-pong buffering. Each buffer is addressed using two 16-bit address fields in the `MLB_CNBCRx` and `MLB_CCBCRx` registers as described below.

- Next Buffer Start Address – `CNBCRx.BSA` – beginning of next buffer in internal memory
- Next Buffer End Address – `CNBCRx.BEA` – ending of next buffer in internal memory
- Buffer Current Address – `CCBCRx.BCA` – beginning of current buffer in internal memory
- Buffer Final Address – `CCBCRx.BFA` – ending of current buffer in internal memory

Data Transfer

For ping-pong DMA mode, transmit and receive for all data types are handled in the following manner.

- At the start of buffer processing, the beginning of the next buffer becomes the beginning of the current buffer, as the BSA bits from the `MLB_CNBCRx` register are loaded into the BCA bit field of the `MLB_CCBCRx` register. Additionally, the end of the next buffer becomes the end of the current buffer, as the BEA bit field from the `MLB_CNBCRx` register is loaded into the BFA bit field of the `MLB_CCBCRx` register.
- A current buffer start interrupt is generated (`STS` bit in the `MLB_CSCRx` register), which informs the software that hardware has updated the `MLB_CCBCRn` register, cleared the local channel `RDY` bit, and is available to accept the next buffer. Software may then prepare the next buffer by writing: `BSA`, `BEA`, and `RDY` bits.
- During the processing of the current buffer, `BCA` bits continue to mark which quadlet of data or packet is currently being processed.
- A current buffer done interrupt is generated when the last quadlet in the current buffer has been successfully transmitted/received.

The current buffer and the next buffer can be configured for either multi-packet or single-packet buffering, when receiving and transmitting asynchronous and control packet data.

- Multi-packet buffering allows the system to reduce the interrupt load at the expense of larger DMA buffers.
- Single-packet buffering allows DMA buffer size to be reduced at the expense of increasing the interrupt rate.

For more information, see “Programming Model” on page 8-16.

Circular Buffer DMA

Logical channels operate in circular buffer DMA mode when the channel mode select bits in the `MLB_CECRx` register are set to 01. This mode is available for synchronous channels only. In contrast to ping-pong buffering, circular buffering uses a single, circular memory buffer to process channel data.

For circular buffer mode, synchronous data is handled in the following manner:

- Before buffer processing can begin, the `BSA` bits in the `MLB_CNBCRx` register and the `BEA` bits in the `MLB_CNBCRx` register should be programmed with the beginning and the ending address of the circular buffer. Set the `RDY` bit in the `MLB_CSCRx` register to initiate buffer processing.
- At the start of buffer processing, the beginning address of the circular buffer (`BSA`) is loaded into `BCA` field of the `MLB_CCBCRx` register. Additionally, the ending address of the circular buffer (`BEA` bits) is loaded into the `BFA` bit field of the `MLB_CCBCRx` register.
- During the processing of the circular buffer, the `BCA` bits are updated to indicate which quadlet of the synchronous data is currently being processed.
- Once the end of the buffer is reached and $BCA = BFA$, the `BCA` field is reloaded to point to the beginning address of the circular buffer (`BSA`).

Unlike in ping-pong DMA, the `RDY` bit remains set during the processing of the circular buffer DMA. Software must clear this bit to halt buffer processing. [For more information, see “Programming Model” on page 8-16.](#)

Interrupts

The buffer start and buffer done interrupts (bit 19 and bit 18 respectively) can unmasked in the channel control registers (MLB_CECRx) to indicate the start and end of a DMA. The MLB interrupt is not selected as one of the 19 programmable interrupts by default. To use the MLB interrupt as one of the 19 programmable interrupts, the appropriate 5 bits in the `PICRx` register have to be programmed to 0x1D. For example, to use MLB interrupt as P18I, the `PICR3` register needs to be programmed to 0x1D.

[Table 8-2](#) provides an overview of MLB interrupts.

Table 8-2. Overview of MLB Interrupts

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
MLB (31channels)	System Status Interrupts -System Detects Channel Scan -Network Unlock -Network Lock -Reset Command -Subcommand -MLB Lock -MLB Unlock -System Service Request Enable	At event detection	-Write 1 to corresponding bit in <code>MLB_SSCR</code> register to clear except for <code>SSRE</code> bit which is cleared by hardware. - RTI instruction	Route MLBI (<code>PICRx</code>) to any <code>PxxI</code>
	Channel Status Interrupts -Receive Service Request -Buffer Done -Transmit Service Request -Buffer Start -Buffer Error -Protocol Error -Detect Break (control and asynchronous only) -Lost Frame Synchronization (synchronous only)		Write 0x0000FFFF to <code>MLB_CSCRx</code> register. This clears corresponding interrupt bit in <code>MLB_CICR</code> register	

Interrupt Source

One interrupt is shared by the system status and channel status interrupts.

Servicing Interrupts

System status interrupts are generated on events that allow system software to monitor and control the status of the MediaLB Network. These interrupts can be unmasked by clearing the corresponding bit in `MLB_SMCR` register. The interrupt is cleared within the ISR before the RTI instruction by writing one to corresponding bit in the `MLB_SSCR` register.

Channel status interrupts are generated on events corresponding to a particular logical channel (0–31). These interrupts can be unmasked by clearing the appropriate bits (16–23) in the `MLB_CECRx` register.

The channel interrupt status register (`MLB_CICR`) reflects the interrupt status of the individual logical channels. For example, if an interrupt is pending in logical channel 0, the corresponding bit (bit 0) is set in the `MLB_CICR` register. These bits are set by hardware when a channel interrupt is generated. The channel interrupt bits are sticky and can only be reset by software. To clear a particular bit in the `MLB_CICR` register, software must clear all of the unmasked status bits in the corresponding `MLB_CSCRn` registers. Therefore, if bit 0 of the `MLB_CICR` register is set, writing 0xFFFF to `MLB_CSCR` bit 0 register clears `MLB_CICR` bit 0. This step must be included the channel interrupt ISR before the RTI instruction to ensure that the interrupt is not regenerated.

Debug Features

The following sections provide information to assist in MediaLB debug.

Programming Model

Loop-Back Test Mode

Loop-back test mode is used for debug operations and is enabled by setting the `LBM` bit in the `MLB_DCCR` register. This mode provides basic testing capabilities for the MediaLB pads, physical layer, link layer, channel protocol and the local channel buffer by enabling a single receive channel and a single transmit channel. When loop-back test mode is enabled, a data path is enabled which allows receive data from even channel N to be sent out as transmit data on channel $N + 1$, where $N = \{0, 2, 4 \dots 30\}$.

Programming Model

The following sections provide procedures that are helpful when programming media local bus interface.

I/O Interrupt Mode

To configure the MLB interface for I/O mode using interrupts, use the following procedure.

1. Reset the MLB device.
2. Program the appropriate bits in the `PICRx` register to generate MLB interrupt.
3. Unmask the appropriate bits in `MLB_SSCR` register inorder to monitor the MLB network.
4. Configure the MLB control register (`MLB_DCCR`) with the appropriate settings and enable the MediaLB device.
5. Check for MLB lock using the status bit in the `MLB_SSCR` register using polling or interrupt.
6. Configure the `MLB_LCBCRx` register for channel buffer threshold, depth and start address.

7. Configure the logical channel using the `MLB_CECRx` register for I/O mode, transfer direction, channel type, channel address and interrupt generation.

For a transmit, a transmit service request, (STS bit 1 in the `MLB_CSCRx` register) an interrupt is generated when the local channel buffer can accept data. Within the ISR, check if this status bit is set. If set, write the data into transmit data buffer (`MLB_CNBCRx`).

For a receive, a receive service request, (STS bit 2 in the `MLB_CSCRx` register) an interrupt is generated when the local channel buffer has data to be read. Within the ISR, check if this status bit is set. If set, read the data from the receive data buffer (`MLB_CCBCRx`).

8. Clear all interrupts by writing 0x0000FFFF to the `MLB_CSCRx` register.

DMA Modes

MLB channels can be configured for circular buffer DMA mode by programming the channel mode select bits (`MLB_CECRx` register, bits 25–26 = 01) for synchronous channels only. In contrast to DMA mode with ping-pong buffering, circular buffering uses a single, circular memory buffer to process channel data.

To configure a ping-pong or circular buffered DMA, use the following procedure.

1. Reset the MLB device.
2. Program the appropriate bits in the `PICRx` register to generate MLB interrupt.
3. Unmask the appropriate bits in the `MLB_SSCR` register in order to monitor the MLB network.

Programming Model

4. Configure the MLB control register (MLB_DCCR) with the appropriate settings and enable the MediaLB device.
5. Configure the base address register (MLB_SBCR, MLB_ABCR or MLB_CBCR) based on the data type configured for the logical channel.
6. Check for MLB lock using the status bit in the MLB_SSCR register using polling or interrupt.
7. Configure the `MLB_LCBCRx` register for channel buffer threshold, depth and start address.
8. Configure the logical channel using the `MLB_CECRx` register for ping-pong or circular buffer DMA mode, transfer direction, channel type, channel address and also to generate appropriate interrupts.
9. Configure the `MLB_CNBCRx` register with the buffer start and end address.
10. Set the RDY bit in the `MLB_CSCRx` register to start the DMA.

Hardware automatically clears the RDY bit for ping-pong DMA but not for circular buffer DMA. Therefore, for circular buffer DMA, this bit should be cleared manually by the software to stop buffer processing.

11. An interrupt is generated depending on the bit unmasked in the `MLB_CECRx` register. Within the ISR check that the appropriate status bit (in the `MLB_CSCRx` register) is set.
12. Clear all interrupts by writing 0x0000FFFF to the `MLB_CSCRx` register.

9 DIGITAL APPLICATION/DIGITAL PERIPHERAL INTERFACES

The digital application interface (DAI) and the digital peripheral interface (DPI) are comprised of a groups of peripherals and their respective signal routing units (SRU and SRU2). The inputs and outputs of the peripherals are not directly connected to external pins. Rather, the SRUs connect the peripherals to a set of pins and to each other, based on a set of configuration registers. This allows the peripherals to be interconnected to suit a wide variety of systems. It also allows the SHARC processors to include an arbitrary number and variety of peripherals while retaining high levels of compatibility without increasing pin count.

The routing unit specifications are listed in [Table 9-1](#).

Table 9-1. Routing Unit Specifications

Feature	DAI	DPI
Pin Buffers		
Number	20	14
Input	Yes	Yes
Output	Yes	Yes
Open-drain	Yes	Yes
Three-state	Yes	Yes
High Impedance	Yes	Yes
Programmable Pull-up	No	No
I/O Level Status Register	Yes	Yes
Interrupts		
Interrupt Control	Yes	Yes

Features

Table 9-1. Routing Unit Specifications (Cont'd)

Feature	DAI	DPI
Total Channels	32	12
Miscellaneous I/O channels	10	9
Peripheral Channels	22	3
Local Memory	No	No
Clock Operation	$f_{PCLK}/4$	$f_{PCLK}/4$

Features

The DAI/DPI incorporates a 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 interface is specified using DAI/DPI registers. For more information on pin states, refer to “[I/O Pin Buffers](#)” on page 9-7.

The DAI/DPI may be used to connect combinations of inputs to combinations of outputs. This function is performed by the SRU/SRU2 via memory-mapped control registers.

This *virtual connectivity* design offers a number of distinct advantages:

- Flexibility
- Increased numbers and kinds of configurations
- Connections can be made via software—no hard wiring is required



Inputs may only be connected to outputs.

Register Overview

The SRU for the DAI contains six register sets that are associated with the DAI groups.

Clock Routing Registers (SRU_CLKx). Associated with Group A, routes clock signals.

Serial Data Routing Registers (SRU_DATx). Associated with group B, routes data.

Frame Sync Routing Control Registers (SRU_FSx). Associated with group C, routes frame syncs or word clocks to the serial ports, the SRC, the S/PDIF, and the IDP.

Pin Signal Assignment Registers (SRU_PINx). Associated with group D, routes physical pins (connected to a bonded pad).

Miscellaneous Signal Routing Registers (SRU_MISCx). Associated with group E, allows programs to route to the DAI interrupt latch, PBEN input routing, or input signal inversion.

DAI Pin Buffer Enable Registers (SRU_PBENx). Associated with group F, Activate the drive buffer for each of the 20 DAI pins.

DAI Shift Registers (SRU_PBENx). Associated with group G, routes all shift register signals.

The SRU2 for DPI contains three register sets associated with the DPI groups.

Miscellaneous Signal Routing Registers (SRU2_INPUTx). Associated with group A, used to route the 14 external pin signals to the inputs of the other peripherals.

Pin Assignment Signal Routing (SRU2_PINx). Associated with group B routes pin output signals to the DPI pins.

Clocking

Pin Enable Signal Routing (SRU2_PBENx). Associated with group C used to specify whether each DPI pin is used as an output or an input by setting the source for the pin buffer enable.

The DAI/DPI registers are unique in that they work as groups to control other peripheral functions. The register groups and routings are described in detail in “DAI/DPI Group Routing” on page 9-18, “DAI Signal Routing Unit Registers” on page A-118 and “DPI Signal Routing Unit Registers” on page A-218.

Clocking

The fundamental timing clock of the DAI/DPI modules is peripheral clock/2 ($\text{PCLK}/2$).

Functional Description

Figure 9-1 shows how the DAI pin buffers are connected via the SRU. This allows for very flexible signal routing.

The DAI/DPI is comprised of four primary blocks:

- Peripherals (A/B/C) associated with DAI/DPI
- Signal Routing Units (SRU, SRU2)
- DAI/DPI I/O pin buffers
- Miscellaneous buffers

The peripherals shown in Figure 9-1 can have up to three connections (if master or slave capable); one acts as signal input, one as signal output and the 3rd as output enable. The SRUs are based on a group of multiplexers which are controlled by registers to establish the desired interconnects.

The DAI/DPI pin buffers have three signals which are used for input/output to/from off-chip world and the 3rd for output enable.

The miscellaneous buffers have an input and an output and are used for group interconnection.

Note that [Figure 9-1](#) is a simplified representation of a DAI system. In a real representation, the SRU and DAI would show several types of data being routed from several sources including the following.

- Serial ports (SPORT)
- Precision clock generators (PCG)
- Input data port (IDP)
- Asynchronous sample rate converters (SRC)
- S/PDIF transmitter
- S/PDIF receiver
- Shift register
- DAI Interrupts (Miscellaneous)

Similarly, the DPI pin buffers are connected via the SRU2. The DPI makes use of several types of data from a large variety of sources, including:

- Peripheral timers
- Serial Peripheral Interfaces (SPI)
- Precision clock generators (PCG)
- Universal asynchronous Rx/Tx ports (UART)
- Two wire interface (TWI)

Functional Description

- GPIO flags (External Port)
- DPI Interrupts (Miscellaneous)

i Note that the precision clock generator (units C/D) can be assigned to access DAI and/or DPI pins.

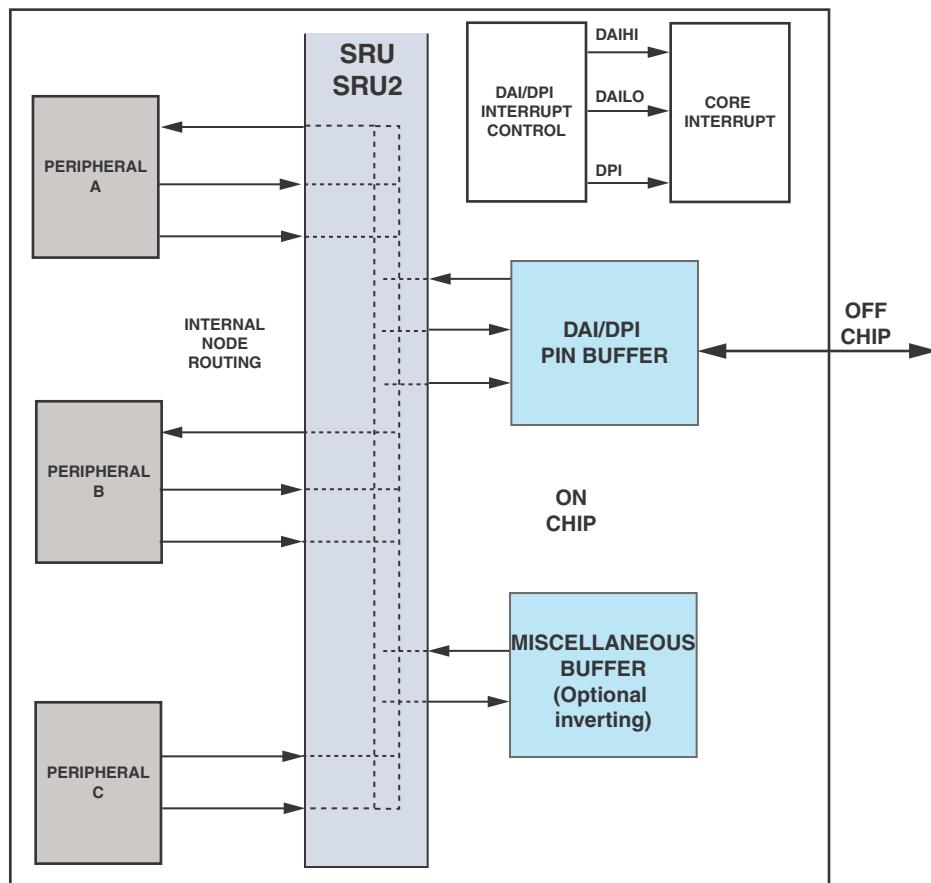


Figure 9-1. DAI/DPI Functional Block Diagram

DAI/DPI Signal Naming Conventions

Each peripheral associated with the DAI/DPI does not have any dedicated I/O pins for off-chip communication. Instead, the I/O pin is only accessible in the chip internally and is known as an *internal node*. Every internal node of a DAI peripheral (input or output) is given a unique mnemonic. The convention is to begin the name with an identifier for the peripheral that the signal is coming to/from followed by the signal's function. A number is included if the DAI contains more than one peripheral type (for example, serial ports), or if the peripheral has more than one signal that performs this function (for example, IDP channels). The mnemonic always ends with *_I* if the signal is an input, or with *_O* if the signal is an output ([Figure 9-2](#)).

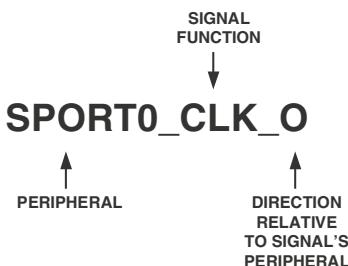


Figure 9-2. Example SRU Mnemonic

I/O Pin Buffers

Within the context of the SRU, physical connections to the DAI pins are replaced by a logical interface known as a *pin buffer*. This is a three terminal active device capable of sourcing/sinking output current when its driver is enabled, and passing external input signals when disabled. Each pin has an input, an output, and an enable as shown in [Figure 9-3](#). 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.

Functional Description

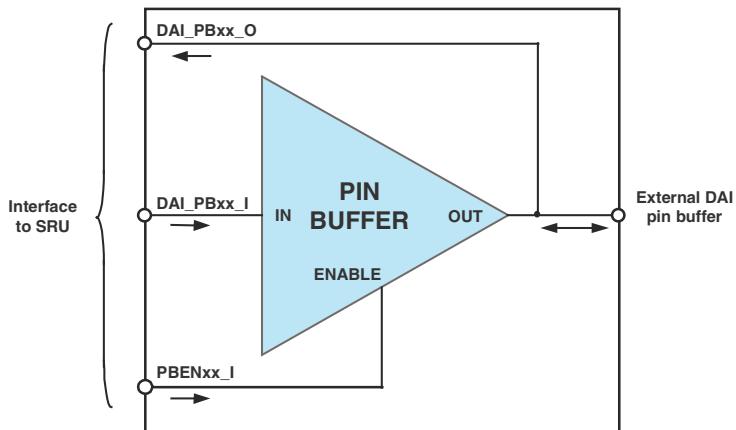


Figure 9-3. Pin Buffer Example

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 routed 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.

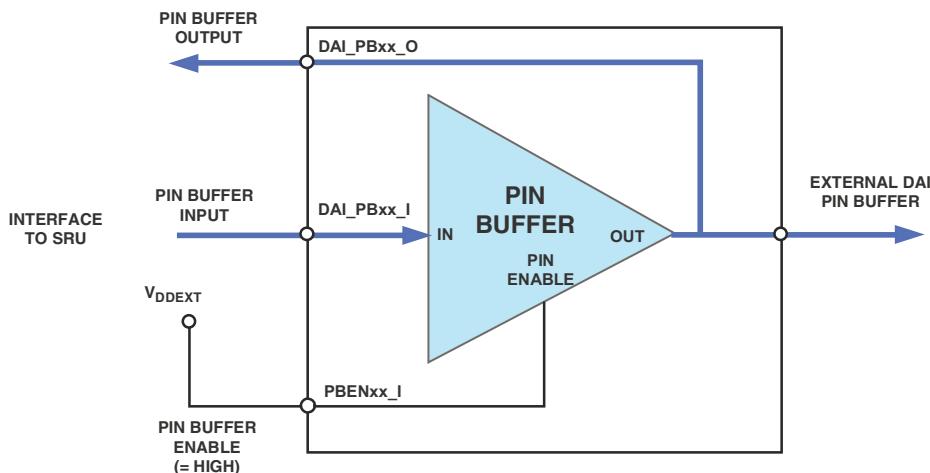
When the pin enable is asserted, the pin output is logically equal to pin input, and the pin is driven. When the pin enable is deasserted, the output of the buffer amplifier becomes high impedance. In this situation, an external device may drive a level onto the line, and the pin is used as an input to the processors.

Pin Buffers as Signal Output

In a typical embedded system, most pins are designated as either inputs or outputs when the circuit is designed, even though they may have the ability to be used in either direction. Each of the DAI pins can be used as

either an output or an input. Although the direction of a DAI pin is set simply by writing to a memory-mapped register, most often the pin's direction is dictated by the designated use of that pin. For example, if the DAI pin were to be hard wired to only the input of another interconnected circuit, it would not make sense for the corresponding pin buffer to be configured as an input. Input pins are commonly tied to logic high or logic low to set the input to a fixed value. Similarly, setting the direction of a DAI pin at system startup by tying the pin buffer enable to a fixed value (either logic high or logic low) is often the simplest and cleanest way to configure the SRU.

When the DAI pin is to be used only as an output, connect the corresponding pin buffer enable to logic high as shown in [Figure 9-4](#). This enables the buffer amplifier to operate as a current source and to drive the value present at the pin buffer input onto the DAI pin and off-chip. When the pin buffer enable ($PBENxx_I$) is set (= 1), the pin buffer output ($PBxx_O$) is the same signal as the pin buffer input ($PBxx_I$), and this signal is driven as an output.



[Figure 9-4. Pin Buffer as Output](#)

Functional Description

Pin Buffers as Signal Input

When the DAI pin is to be used only as an input, connect the corresponding pin buffer enable to logic low as shown in [Figure 9-5](#). This disables the buffer amplifier and allows an off-chip source to drive the value present on the DAI pin and at the pin buffer output. When the pin buffer enable ($PBENxx_I$) is cleared (= 0), the pin buffer output ($PBxx_O$) is the signal driven onto the DAI pin by an external source, and the pin buffer input ($PBxx_I$) is not used.



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 ([Figure 9-5](#) and [Figure 9-6](#)).

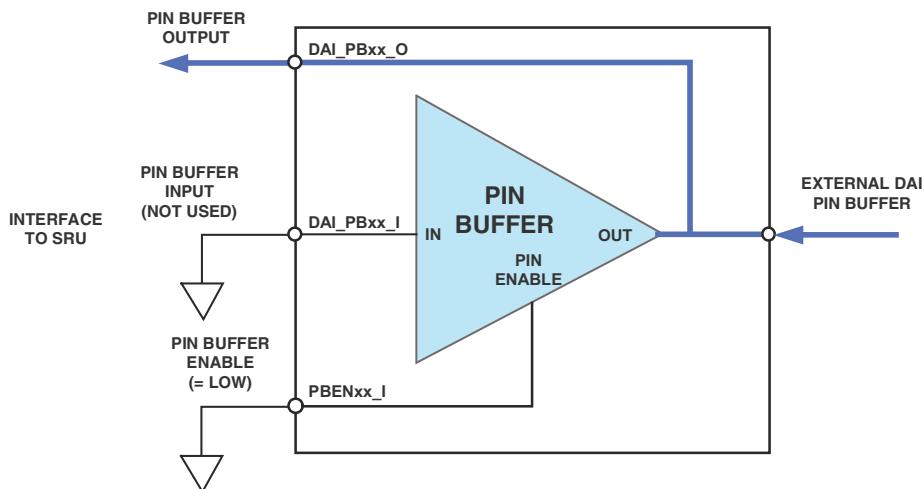


Figure 9-5. Pin Buffer as Input

By default, some pin buffer enables are connected to SPORT pin enable signals that may change value. Tying the pin buffer input low decouples the line from irrelevant signals and can make code simpler to debug. It also ensures that no voltage is driven by the pin if a bug in your code accidentally asserts the pin enable.

Pin Buffers as Open Drain

For peripherals like the TWI and SPI (multi processing), the bus protocol requires the pin drivers to work in open drain mode (Figure 9-6) for transmit and receive operation. The signal input of the assigned pin buffer is tied low. The peripheral's data output signal is connected to the `PBEN` signal. In open drain mode, if `PBEN` = low, the level on the pin is depending on the bus activities. If `PBEN` = high, the driver is conducting (input always low level) and ties the bus low level. Note that for the SPI the `ODP` bit in the `SPICTL` register must be enabled.

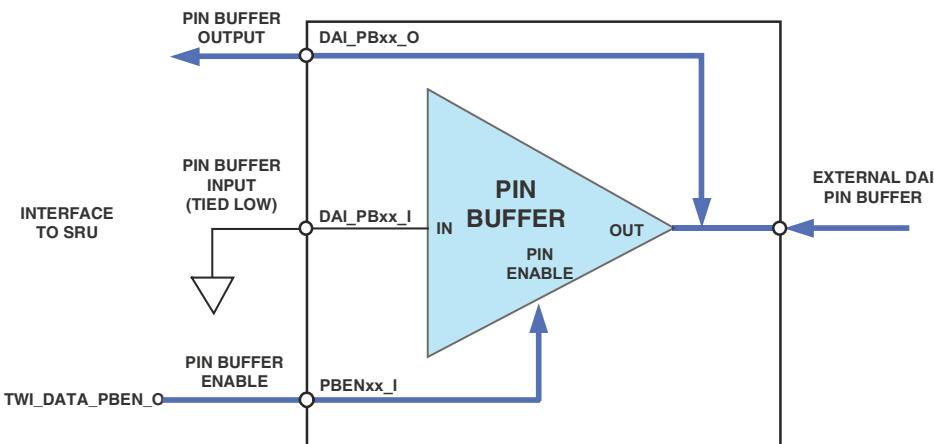


Figure 9-6. Pin Buffer as Open Drain



In open drain mode the data output signal is the peripheral's `PBEN_0` signal

DAI/DPI Pin Buffer Status

The signal levels on the DAI/DPI pins can be read with the `DAI/DPI_PIN_STAT` registers. This allows conditions like for example:

Functional Description

```
ustat2=dm(DAI_PIN_STAT);  
bit tst ustat2 DAI_PB10;  
if TF jump DAI_PB10_high;
```

Unused DAI/DPI Pins

If a DAI/DPI pin is not being used, its pin enable (for example `DAI_PBENxx_I`) and its input (`DAI_PBxx_I`) for its pin buffer should be connected to low.

Miscellaneous Buffers

The miscellaneous buffers (Figure 9-7) are used to interconnect signals from different routing groups. These buffers have the following characteristics.

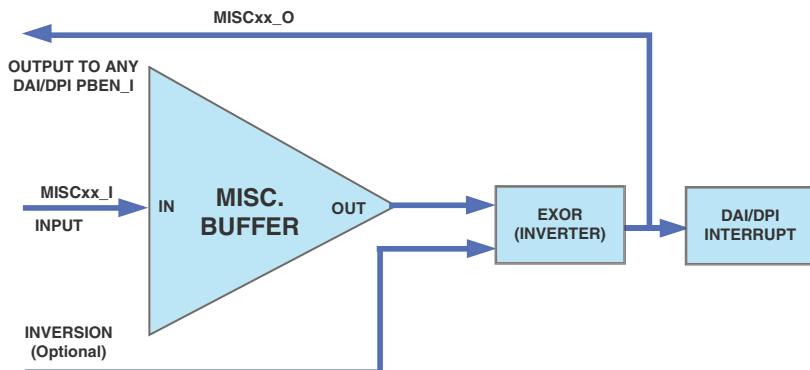


Figure 9-7. Miscellaneous Buffer

1. Only for internal connections, no pin buffer enable required.
2. `MISCxx_O` output always feeds DAI/DPI interrupt latch register and group F (DAI) and group C (DPI) to route to any DAI/DPI `PBEN_I` signal.

3. MISCxx_I input sources collected from different groups.
4. Some buffers allow signal inversion.

The miscellaneous buffers acts as intermediate buffer connections between the peripheral's source node and the pin buffer enable destination node. This allows for routing that are not possible among a single group.

 The miscellaneous buffer allows interconnects which are not supported within a DAI/DPI routing group.

[Table 9-2](#) and [Table 9-3](#) provide routing options for the DAI and DPI miscellaneous buffers. Note that while the Interrupt and Miscellaneous Inputs have different naming conventions, they are the same physical input field.

Table 9-2. Miscellaneous DAI Buffer Routing

Register	Inputs		Outputs		
	Interrupt	Miscellaneous	Interrupt Trigger	Signal Inversion	DAI_PBENxx_I Routing
SRU_EXT_MISCA	DAI_INT_28_I	MISCA0_I	Yes	No	Yes
	DAI_INT_29_I	MISCA1_I	Yes	No	Yes
	DAI_INT_30_I	MISCA2_I	Yes	No	Yes
	DAI_INT_31_I	MISCA3_I	Yes	No	Yes
		MISCA4_I	No	Yes	Yes
		MISCA5_I	No	Yes	Yes
SRU_EXT_MISCB	DAI_INT_22_I		Yes	No	Yes
	DAI_INT_23_I		Yes	No	Yes
	DAI_INT_24_I		Yes	No	Yes
	DAI_INT_25_I		Yes	No	Yes
	DAI_INT_26_I		Yes	No	Yes
	DAI_INT_27_I		Yes	No	Yes

Functional Description

Table 9-3. Miscellaneous DPI Buffer Routing

Register	Inputs		Outputs		
	Interrupt	Miscellaneous	Interrupt Trigger	Signal Inversion	DPI_PBENxx_I Routing
SRU2_INPUT4	DPI_INT_05_I	MISCB0_I	Yes	No	Yes
	DPI_INT_06_I	MISCB1_I	Yes	No	Yes
	DPI_INT_07_I	MISCB2_I	Yes	No	Yes
SRU2_INPUT5	DPI_INT_08_I	MISCB3_I	Yes	No	Yes
	DPI_INT_09_I	MISCB4_I	Yes	No	Yes
	DPI_INT_10_I	MISCB5_I	Yes	No	Yes
	DPI_INT_11_I	MISCB6_I	Yes	No	Yes
	DPI_INT_12_I	MISCB7_I	Yes	No	Yes
	DPI_INT_13_I	MISCB8_I	Yes	No	Yes

DAI/DPI Peripherals

There are two categories of peripherals associated with the DAI and DPI. These are described in the following sections.

Output Signals With Pin Buffer Enable Control

Many peripherals within the DAI/DPI that have bidirectional pins generate a corresponding pin enable signal. Typically, the settings within a peripheral's control registers determine if a bidirectional pin is an input or an output, and is then driven accordingly.

 Both the peripheral control registers and the configuration of the SRU can effect the direction of signal flow in a pin buffer.

from an external perspective for example, when a serial port (SPORT) is completely routed off-chip, it uses four pins—clock, frame sync, data channel A, and data channel B. Because all four of these pins comprise the interface that the serial port presents to the SRU, there are a total of 12 connections as shown in [Figure 9-8](#).

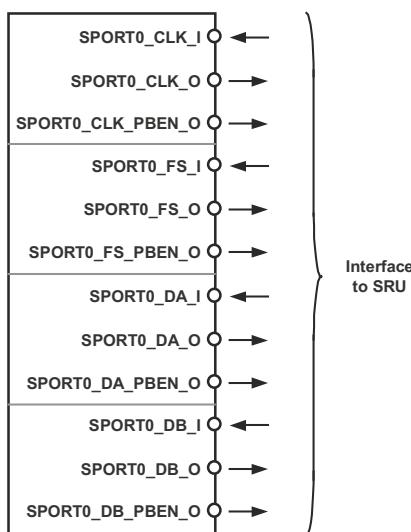


Figure 9-8. SRU Connections for SPORT0

For each bidirectional line, the SPORT provides three separate signals. For example, a SPORT clock has three separate SRU connections (instead of one physical pin):

- input clock to the SPORT (`SPORTx_CLK_I`)
- output clock of the SPORT (`SPORTx_CLK_O`)
- output enable clock of the SPORT (`SPORTx_CLK_PBEN_O`)

If the SPORT operates in master mode, the `SPORTx_CLK_O` and `SPORTx_CLK_PBEN` signals are automatically driven. If operating in slave mode, the `SPORTx_CLK_O` and `SPORTx_CLK_PBEN` signals are automatically disabled and the `SPORTx_CLK_I` signal expects an external clock.



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 the SRU, the standard SPORT configuration registers

Functional Description

behave as documented in [Chapter 10, Serial Ports](#). The SRU then becomes transparent to the peripheral. [Figure 9-8](#) demonstrates SPORT0 properly routed to DAI pins 1 through 4; although it can be equally well routed to any of the 20 DAI pins.

The pin buffer output enable signals (x_PBEN_0) supported by any peripheral should always be routed to the dedicated pin buffer enable input (DAI_PBEN_I, DPI_PBEN_I).

Output Signals Without Pin Buffer Enable Control

Some peripherals have signal outputs without automated pin buffer control enable (PDAP_STRB_0, MISCx_0, BLK_START_0).

The operation of these peripherals is simplified as the routing to a DAI/DPI pin buffer enable input requires a static high from the SRU. In order to disable the pin buffer output, software must clear the pin buffer enable input accordingly.

Signal Routing Units (SRUs)

The following sections provide more detail specific to the SRUs.

Signal Routing Matrix by Groups

The SRU can be likened to a set of patch bays, which contains a bank of inputs and a bank of outputs. For each input, 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. With the grouping, the multiplexing scheme becomes highly efficient since it wouldn't make sense for instance to route a frame sync signal to a data signal.

The SRU for the DAI contains seven groups that are named sequentially A through F. Each group routes a unique set of signals with a specific purpose as shown below.

- Group A routes clock signals
- Group B routes serial data signals
- Group C routes frame sync signals
- Group D routes pin signals
- Group E routes miscellaneous signals
- Group F routes pin output enable signals
- Group G routes all shift register signals (ADSP-2147x only)

Together, the SRU's seven groups include all of the inputs and outputs of the DAI peripherals, a number of additional signals from the core, and all of the connections to the DAI pins.

The SRU2 for DPI contains three groups that are named sequentially A through C. Each group routes various signals with a specific purpose:

- Group A routes miscellaneous signals
- Group B routes pin output signals
- Group C routes pin output enable signals



Unlike SRU in the DAI module, all types of functionality such as clock and data are merged into the same group in the DPI peripheral.

Note that it is not possible to connect a signal in one group directly to signal in a different group (analogous to wiring from one patch bay to another). However, group D (DAI) or group B (DPI) is largely devoted to routing in this vein.

Functional Description

DAI/DPI Group Routing

Each group has a unique encoding for its associated output signals and a set of configuration registers. For example, DAI group A is used to route clock signals. The memory-mapped registers, SRU_CLKx, contain bit fields corresponding to the clock inputs of various peripherals. The values written to these bit fields specify a signal source that is an output from another peripheral. All of the possible encodings represent sources that are clock signals (or at least could be clock signals in some systems). [Figure 9-9](#) diagrams the input signals that are controlled by the group A register, SRU_CLKx. All bit fields in the SRU configuration registers correspond to inputs. The value written to the bit field specifies the signal source. This value is also an output from some other component within the SRU.

The SRU is similar to a set of patch bays. Each bay routes a distinct set of outputs to compatible inputs. These connections are implemented as a set of memory-mapped registers with a bit field for each input. The outputs are implemented as a set of bit encodings. Conceptually, a patch cord is used to connect an output to an input. In the SRU, a bit pattern that is associated with a signal output (shown in [Figure 9-9](#)) is written to a bit field corresponding to a signal input.

The same encoding can be written to any number of bit fields in the same group. It is not possible to run out of patch points for an output signal.

Just as group A routes clock signals, each of the other groups route a collection of compatible signals. Group B routes serial data streams while group C routes frame sync signals. 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, but group D is required to route a signal to the pin input. Group F routes signals to the pin enables, and the value of these signals determines if a DAI pin is used as an output or an input. These groups are described in more detail in the following sections.

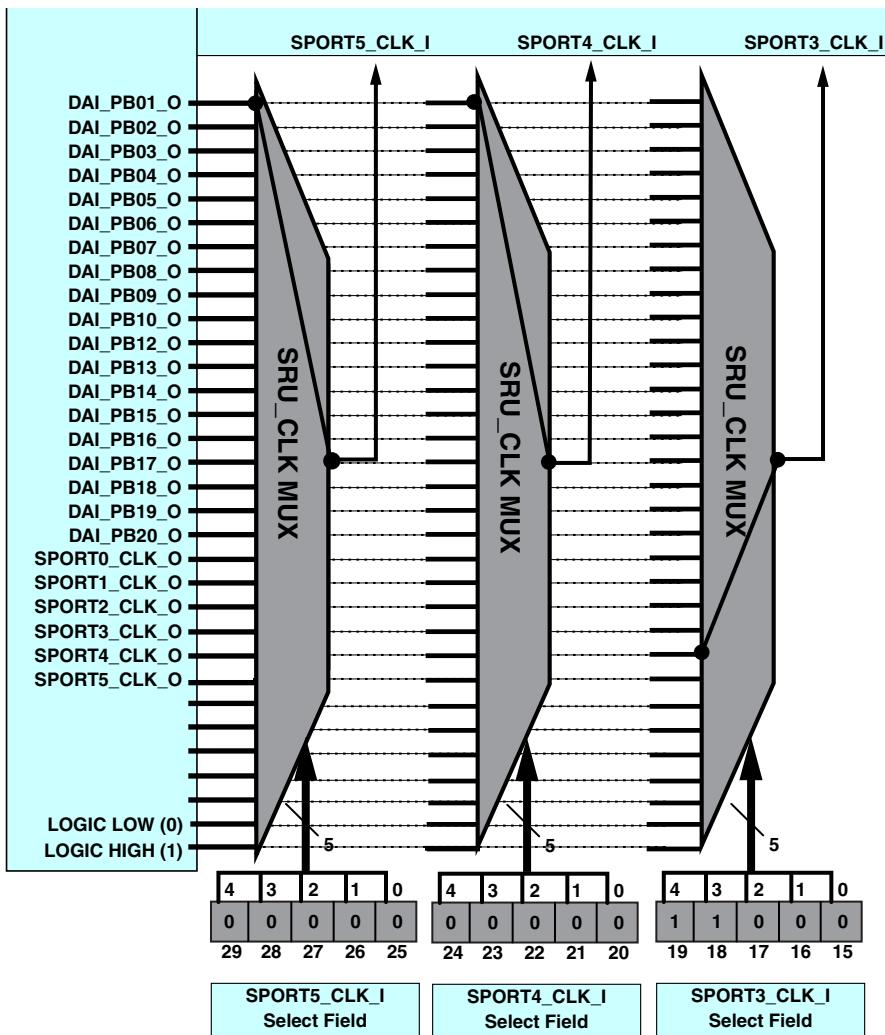


Figure 9-9. Example DAI SRU Group A Multiplexing (SRU_CLKx)

Functional Description

Rules for SRU Connections

There are two rules which apply to all routing:

1. Each input must connect to exactly one output
2. An output can feed any number of inputs

As an example from [Figure 9-9](#):

- DAI_PB01_0 is routed to SPORT5_CLK_I
- DAI_PB01_0 is routed to SPORT4_CLK_I
- SPORT4_CLK_0 is routed to SPORT3_CLK_I



Inputs may only be connected to outputs.

Making SRU Connections

In this section, three types of SRU routing are demonstrated.

1. [Listing 9-1](#) and [Figure 9-10](#) show the SRU connection between the DAI and pin buffers.
2. [Listing 9-2 on page 9-22](#) and [Figure 9-11 on page 9-22](#) show the SRU connection between the DAI pin buffers and SPORTs.
3. [Listing 9-3 on page 9-23](#) and [Figure 9-12 on page 9-23](#) show SRU connection from the SPORT/PCG to the MISC/DAI pin buffers.



These examples use a macro which is provided by the VisualDSP tools. Also see “[Programming Model](#)” on [page 9-42](#).

[Listing 9-1. SRU Connection Between DAI Pin Buffers](#)

```
SRU(HIGH, PBEN03_I);      // DAI pin 3 output
nop;
SRU(LOW, PBEN14_I);       // DAI pin 14 input
```

Digital Application/Digital Peripheral Interfaces

```
nop;  
SRU(LOW, DAI_PB14_I); // DAI pin 14 input level low  
nop;  
SRU(DAI_PB14_O, DAI_PB03_I); // connect DAI pin 14 to DAI pin 3  
nop;  
SRU(DAI_PB14_O, DAI_INT_22_I); // connect DAI pin 14 to DAI  
interrupt 22
```

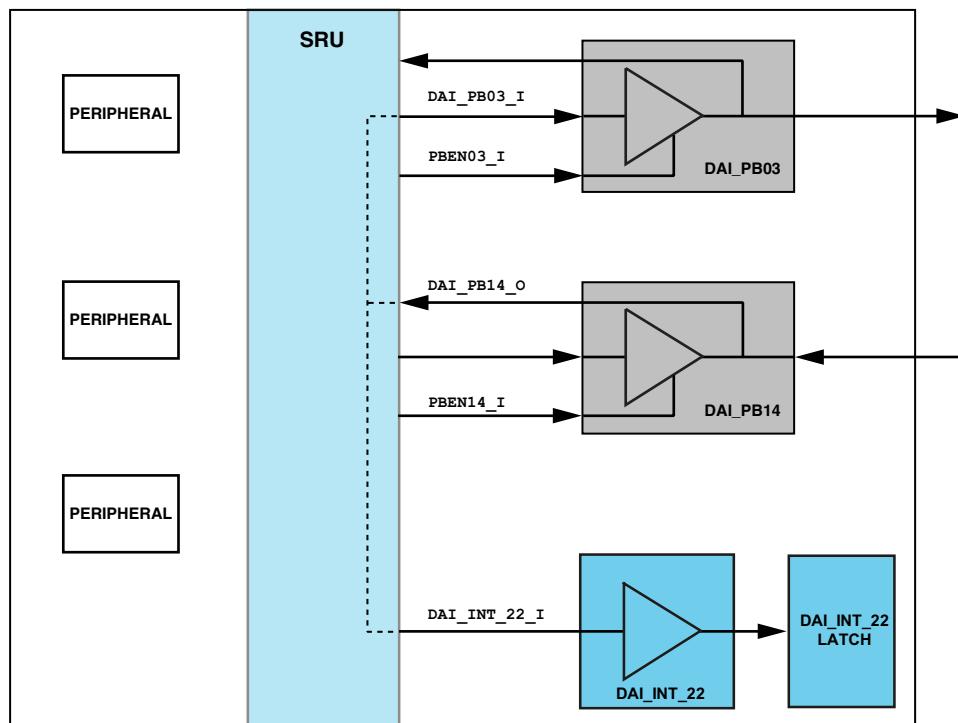


Figure 9-10. SRU Connection Between DAI Pin Buffers

Functional Description

Listing 9-2. SRU Connection Between DAI Pin Buffers and SPORTS

```
SRU(SPORT0_CLK_PBEN_0, PBEN03_I); // DAI pin 3 as output  
nop;  
SRU(SPORT0_CLK_0, DAI_PB03_I); // connect to DAI pin 3  
nop;  
SRU(SPORT0_CLK_0, SPORT1_CLK_I); // connect to SPORT1  
nop;  
SRU(SPORT0_CLK_0, SPORT2_CLK_I); // connect to SPORT2
```

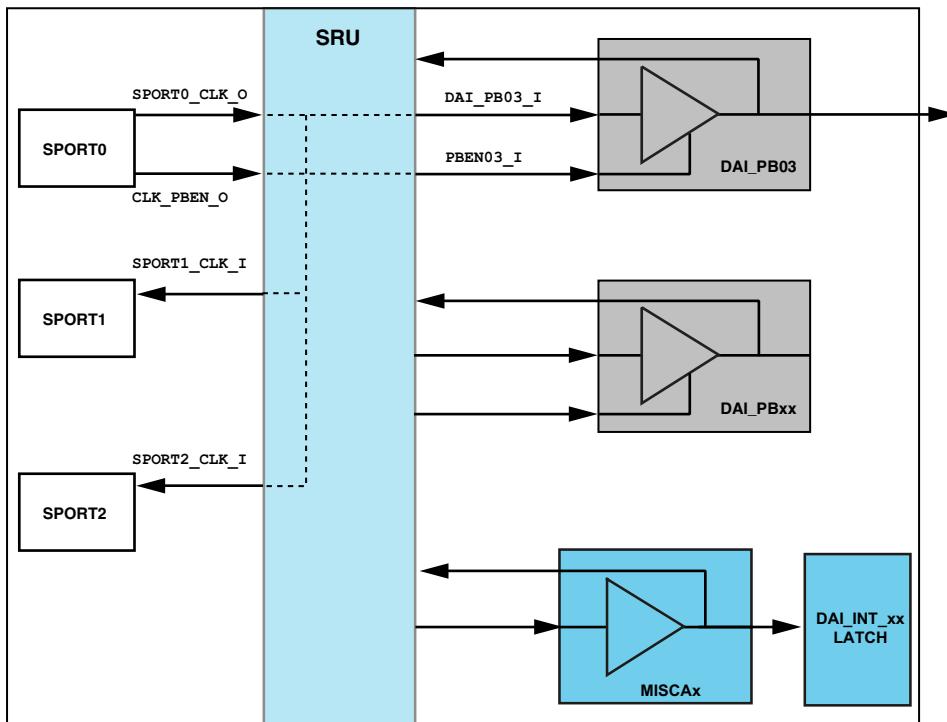


Figure 9-11. SRU Connection Between DAI Pin Buffers and SPORTs

Listing 9-3. SRU Connection SPORT/PCG to MISC/DAI Pin Buffers

```

SRU(HIGH, PBEN03_I);           // DAI pin 3 output
nop;
SRU(DAI_PB14_O, DAI_PB03_I);   // connect pin 3 and 14
nop;
SRU(PCG_CLKB_0, DAI_PB14_I);   // connect PCG and pin 14
nop;
SRU(SPORT2_FS_0, MISCA4_I);    // connect SPORT to MISCA
nop;
SRU(MISCA4_O, PBEN14_I);      // connect MISCA to PBEN14
nop;
SRU(HIGH, INV_MISCA4_I);       // invert MISCA4 input

```

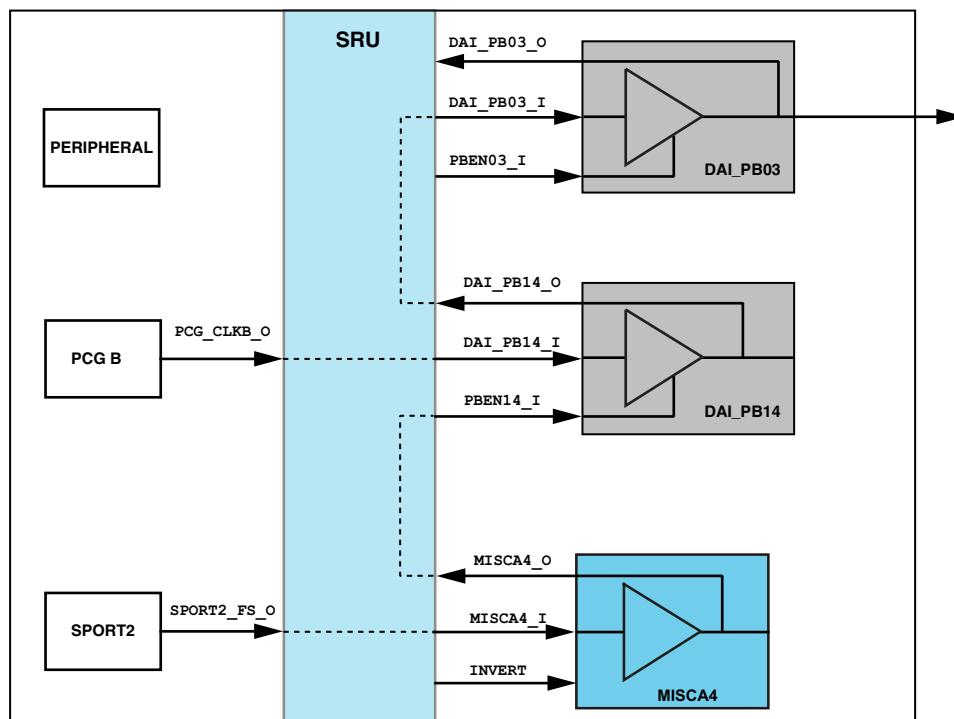


Figure 9-12. SRU Connection SPORT/PCG to MISC/DAI Pin Buffers

Functional Description

DAI Routing Capabilities

[Table 9-1](#) provides an overview about the different routing capabilities for the DAI unit. The DAI groups allow routing of specific signals like clocks, data, frame syncs.

Table 9-4. DAI Routing Capabilities

DAI Group	Input (xxxx_I)	Output (xxxx_O)	
A-Clocks	SPORT7-0 SRC3-0 IDP7-0 PCG A, B, C, D (Ext. clock, Ext. Sync) S/PDIF-Tx (clock, HF Clock, ext. sync) SPDIF-Rx (ext. clock)	SPORT5-0 PCG A, B S/PDIF Rx (clock, TDM clock)	DAI Pin Buffer20-1 Logic level high Logic level low
B-Data	SPORT7-0 A, B SRC3-0 (data, TDM data) IDP7-0 S/PDIF Tx/Rx	SPORT7-0 A, B SRC3-0(data, TDM data) S/PDIF Tx/Rx	
C-Frame Sync	SPORT7-0 SRC3-0 IDP7-0	SPORT5-0 PCG A, B S/PDIF RxX	
D-Pin Buffer Inputs	DAI Pin Buffer 20-1 DAI Pin Buffer 19 Inversion DAI Pin Buffer 20 Inversion	SPORT7-0A/B (data) SPORT7-0 (clock, FS, TDV, data) S/PDIF Rx (clock, TDM clock, FS, data, ext. PLL) S/PDIF Tx (data, block start) PDAP (output strobe) PCG C, D (clock, FS) SRC3-0 (data)	
E-Miscellaneous Signals	DAI Interrupt 31-22 MISCA5-0 MISCA4 Input Inversion MISCA5 Input Inversion	SPORT5-0 (FS) PCG A (clock) PCG B (clock, FS) S/PDIF Tx (block start)	

Table 9-4. DAI Routing Capabilities

DAI Group	Input (xxxx_I)	Output (xxxx_O)	
F-Pin Buffer	DAI Pin Buffer Enable 20–1	SPORT7–0 (clock, FS, data, TDV) MISCA5–0	Logic level high Logic level low
G-Shift Register (ADSP-2147x only)	SR_CLK_I SR_LAT_I SR_SDI_I	SPORT7–0 (clk, FS) SPORT7–0 AB (data) PCG A-B (clk, FS) DAI Pin Buffer 8–1 SR_CLK, SR_LAT, SR_SDI	

DPI Routing Capabilities

[Table 9-2](#) provides an overview about the different routing capabilities for the DPI unit.

Table 9-5. DPI Routing Capabilities

DPI Group	Input (xxxx_I)	Output (xxxx_O)	
A–Miscellaneous Signals	SPI (MOSI, MISO, DS, CLK) SPIB (MOSI, MISO, DS, CLK) TWI (Clock, Data) UART0 RX data Timer1–0 FLAG15–4/PWM3–1 MISCB8–0 DPI Interrupt 13–5	Timer1–0 UART0 TX Data	DPI Pin Buffer Logic level high Logic level low
B–Pin Buffer Input	DPI Pin Buffer Input	Timer1–0 UART0 TX data SPI (MOSI, MISO, DS, CLK, SPIFLG) SPIB (MOSI, MISO, DS, CLK, SPIBFLG) FLAG15–4/PWM3–1 PCG (C, D) (clock, FS)	

Functional Description

Table 9-5. DPI Routing Capabilities (Cont'd)

DPI Group	Input (xxxx_I)	Output (xxxx_O)	
C-Pin Buffer Enable	DPI Pin Buffer Enable	Timer1–0 SPI (MOSI, MISO, DS, CLK) SPIB (MOSI, MISO, DS, CLK) UART0 TX FLAG15–4 TWI (clock, data) MISCB8–0	Logic level high Logic level low

Pin Buffer Input

DAI group D or DPI group B are used to specify any signals that are driven off-chip by the pin buffers. A pin buffer input ($PBxx_I$) is driven as an output from the processor when the pin buffer enable is set (= 1).

Each physical pin (connected to a bonded pad) may be connected via the SRU to any of the outputs of the DAI/DPI peripherals, based on the bit field values. The SRU also may be used to route signals that control the pins in other ways. Many signals may be configured for use as control signals.

Any of the DAI/DPI 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 to assert signals. They can also be used as DAI/DPI interrupt sources.

On the DAI, two dedicated input pin buffers are allowed to invert the input level on the pins by a bit setting. However this only applies if the buffer is not assigned to itself.

Pin Buffer Enable

DAI group F or DPI group C signals are used to specify whether each DAI/DPI pin is used as an output or an input by setting the source for the pin buffer enables. When a pin buffer enable ($PBENxx_I$) is set (= 1), the

signal present at the corresponding pin buffer input (`PBxx_I`) is driven off-chip as an output. When a pin buffer enable is cleared (= 0), the signal present at the corresponding pin buffer input is ignored.

The pin enable control registers activate the drive buffer for each of the DAI/DPI pins. When the pins are not enabled (driven), they can be used as inputs.

Though peripherals are capable of operating bi-directionally, it is not required that all peripheral's `_I` and `_O` signals should be connected to the pin buffer. If the system design only uses a signal in one direction, it is simpler to connect the pin buffer enable pin directly to high or low as appropriate.

Furthermore, signals in the SRU other than the pin buffer enable signal (which is generated by the peripheral) may be routed to the pin buffer enable input. For example, an outside source may be used to 'gate' a pin buffer output by controlling the corresponding pin buffer enable.

Miscellaneous Signals

DAI group E or DPI group C connections are slightly different from the others in that the inputs and outputs being routed vary considerably in function. This group routes control signals and provides a means of connecting signals between groups.

For the DAI, the `MISCAx_I` signals appear as inputs in group E, but do not directly feed any peripheral. Rather, the `MISCAx_O` signals reappear as outputs in group F.

For the DPI, the `MISCBx_I` signals appear as inputs in group A, but do not directly feed any peripheral. Rather, the `MISCBx_O` signals reappear as outputs in group C.

Additional connections among groups provide a surprising amount of utility. Since the output groups F and C dictate pin direction, these few

DAI Default Routing

signal paths enable a number of possible uses and connections for the DAI/DPI 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.

On the DAI, two dedicated miscellaneous inputs are allowed to invert the input level on the buffer by a bit setting.

The SRU enables many possible functional changes, both within the processor as well as externally. Used creatively, it allows system designers to radically change functionality at runtime, and to potentially reuse circuit boards across many products.

DAI Default Routing

When the processor comes out of reset, the SPORT junctions are bi-directional to the DAI pin buffers ([Figure 9-13](#), [Figure 9-14](#)). This allows systems to use the SPORTs as either master or slave (without changing the routing scheme). Therefore, programs only need to use the SPORT control register settings to configure master or slave operation. Note that all DAI inputs which are not routed by default are tied to signal low.

Digital Application/Digital Peripheral Interfaces

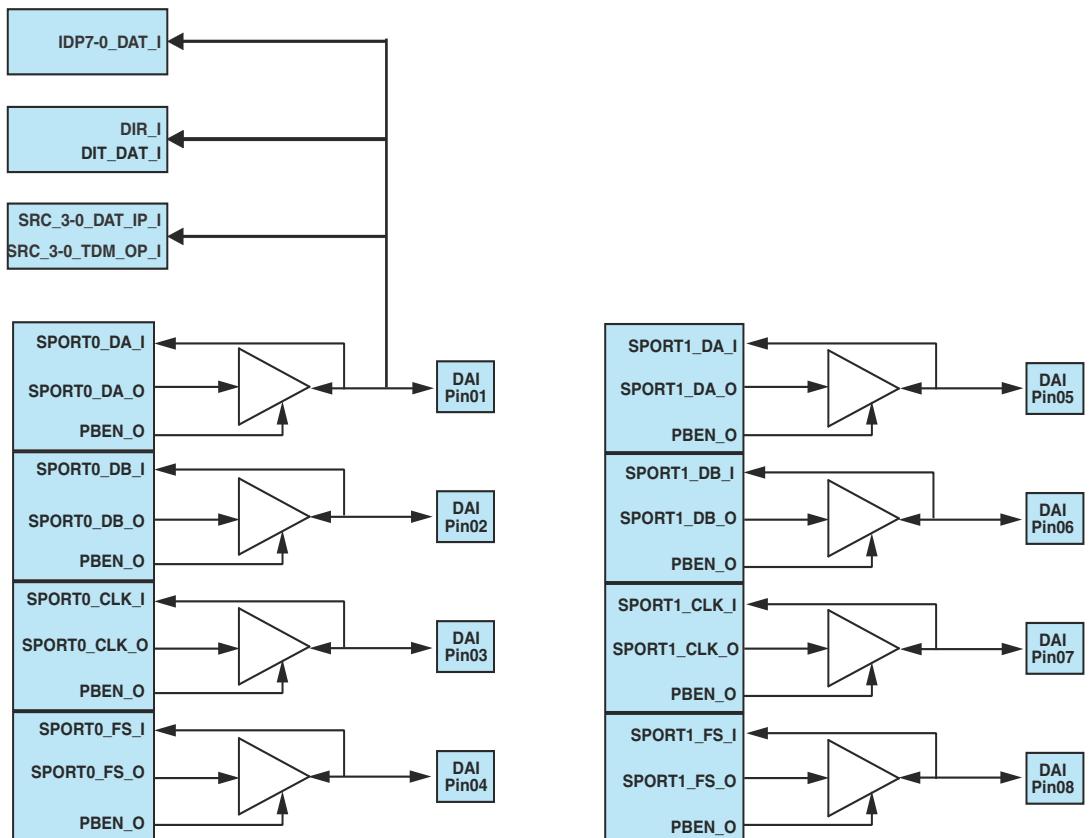


Figure 9-13. DAI Default Routing

DAI Default Routing

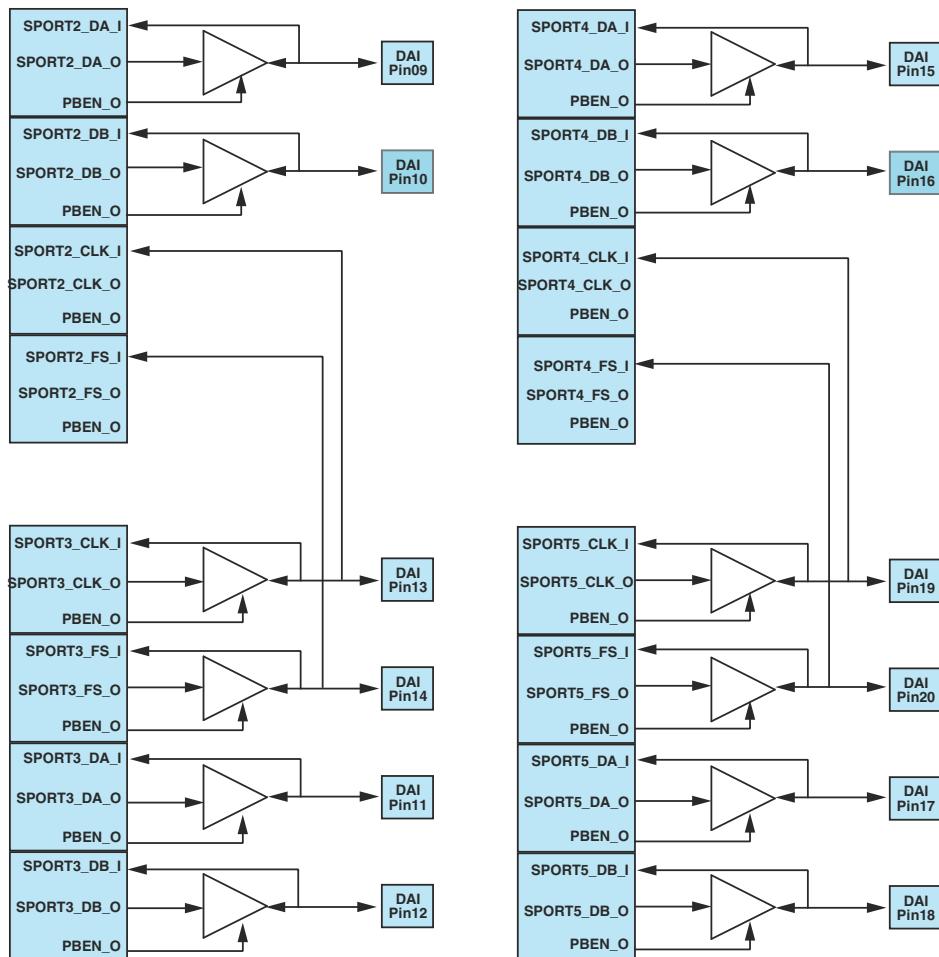


Figure 9-14. DAI Default Routing (Con't)

DPI Default Routing

When the processor comes out of reset, some default routing is established (Figure 9-15). This scheme allows systems to use the SPI as either master or slave (without changing the routing scheme). Programs only need to use the SPI control register settings to configure master or slave operation.

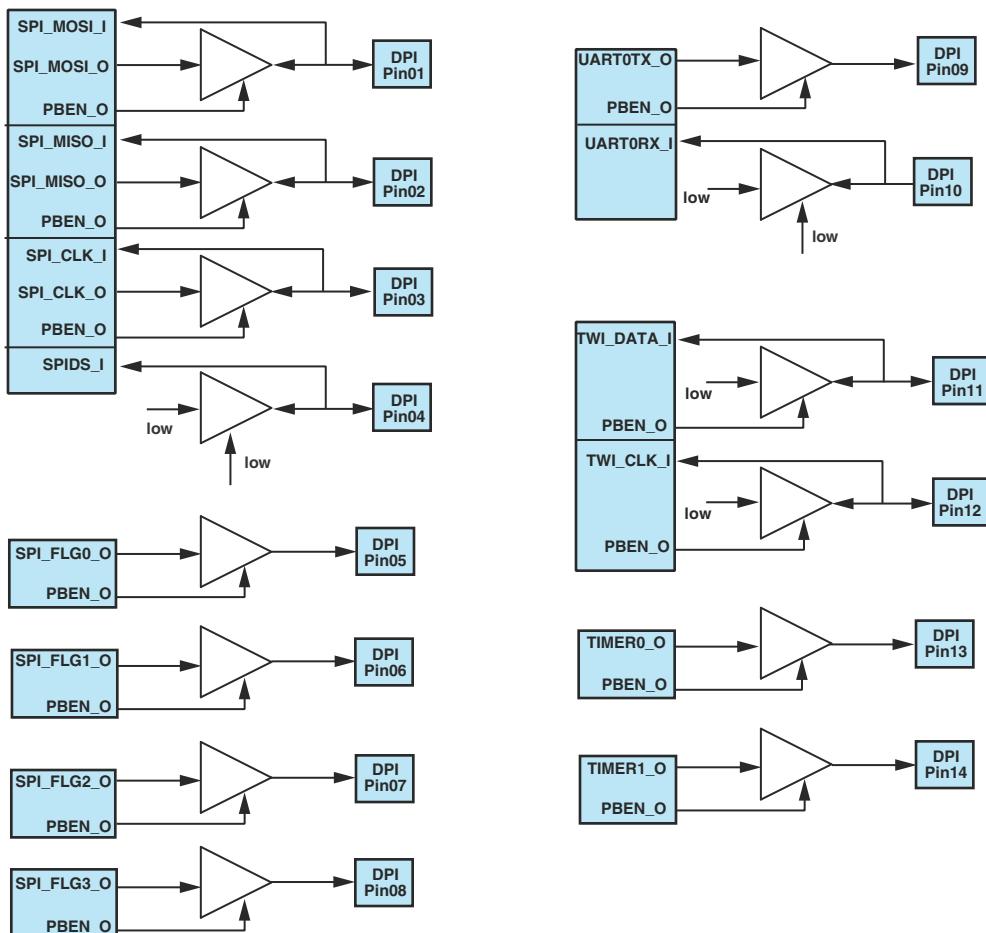


Figure 9-15. DPI Default Routing

Interrupts



All DPI inputs which are not routed by default are tied to signal low. The default routing is used for SPI master/slave booting.

Interrupts

The DAI contains a dedicated interrupt controller that signals the core when DAI peripheral events occur.

System Versus Exception Interrupts

Generally, interrupts are classified as system or exception. Exception events include any hardware interrupts (for example, resets) and emulation interrupts, math exceptions, and illegal accesses to memory that does not exist.

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).

Exception events are treated as high priority events. In comparison, system interrupts are “deterministic”—specific events emanating from a source (the causes), the result of which is the generation of an interrupt. The expiration of a timer can generate an interrupt, a signal that a serial port has received data that must be processed, a signal that an SPI has either transmitted or received data, and other software interrupts like the insertion of a trap that causes a breakpoint—all are conditions, which identify to the core that an event has occurred.

Since DAI specific events generally occur infrequently, the DAI IC classifies such interrupts as either high or low priority interrupts. Within these

broad categories, programs can indicate which interrupts are high and which are classified as low.

Functional Description

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.

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 processor core, the DAI interrupt controller reduces all of its interrupts onto two interrupt signals within the core's primary interrupt systems.

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.

DAI Interrupt Channels

The DAI can handle up to 32 interrupts as shown below.

- 8 x IDP DMA channels (Input data port)
- 2 x IDP FIFO status (Input data port)
- 10 x miscellaneous interrupts
- 8 x S/PDIF receiver status
- 4 x SRC (sample rate converter)

DAI Interrupt Priorities

As described above, the DAI interrupt controller registers provide 32 independently-configurable interrupts labeled `DAI_INT31-0`.

Just as the core has its own interrupt latch registers (`IRPTL` and `LIRPTL`), the DAI has its own latch registers (`DAI_IMASK_L` and `DAI_IMASK_H`). When a DAI interrupt is configured to be high priority, it is latched in the `DAI_IMASK_H` register. When any bit in the `DAI_IMASK_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_IMASK_L` register. Similarly, when any bit in the `DAI_IMASK_L` register is set (= 1), bit 6 in the `LIRPTL` register is also set and the core services that interrupt with low priority.



By default interrupts are mapped onto low priority interrupt.

DPI Interrupt Channels

The DPI can handle up to 12 interrupts as shown below.

- 1 TWI FIFO status
- 2 UART DMA channels
- 9 miscellaneous interrupts

DPI Interrupt Priorities

Just as the core has its own interrupt latch registers (`IRPTL` and `LIRPTL`), the DPI has its own latch registers (`DPI_IMASK`). When a DPI interrupt is configured, it is latched in the `DPI_IMASK` register. When any bit in the `DPI_IMASK` register is set (= 1), bit 11 (DPII) in the `IRPTL` register is also set and the core services that interrupt.



The DPI interrupt controller has no priority option.

DAI Miscellaneous Interrupts

As described above, the DAI interrupt controller registers provide 10 independently-configurable interrupts labeled as `DAI_31-22_INT`. Any trigger on the routed DAI inputs `DAI_INT_31-22_I` can cause an interrupt latch event in `DAI_IMASK` register if unmasked.



There are two signal naming conventions: the DAI interrupt controller bits are named `DAI_31-22_INT` and its corresponding SRU signals are named `DAI_INT_31-22_I`.

Signals from the SRU can be used to generate interrupts. For example, when `DAI_30_INT` (bit 30) of `DAI_IMASK_H` is set to one, any signals from the external miscellaneous channel A2 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.

[Table 9-6](#) provides an overview of DAI miscellaneous interrupts.

Table 9-6. DAI Miscellaneous Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
DAI MISCA (10 channels)	<ul style="list-style-type: none"> – Rising edge – Falling edge – Rising/falling edge 		Read-to-clear <code>DAI_IRPTL_x</code> + RTI instruction	P0I, P12I

DPI Miscellaneous Interrupts

As described above, the DPI interrupt controller registers provide 9 independently-configurable interrupts labeled as `DPI_13-5_INT`. Any trigger on the routed DPI inputs `MISCB8-0_I` can cause an interrupt latch event in `DPI_13-5_INT` if enabled.

Interrupts



There are two signal naming conventions: the DPI interrupt controller bits are named `DPI_13-5_INT` and its corresponding SRU signals are named `MISCB8-0_I`.

Signals from the SRU2 group C can be used to generate interrupts. For example, when `DPI_13_INT` (bit 13) of `DPI_IMASK` is set to one, any signals from the miscellaneous channel 8 `MISCB8_I` generates an interrupt. If set to one, the DPI triggers 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 DPI interrupt indicates the source (in this case, miscellaneous, Channel 8), and checks the IVT for an instruction (next operation) to perform.

[Table 9-7](#) provides an overview of DPI miscellaneous interrupts.

Table 9-7. DPI Miscellaneous Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
DPI MISCB (9 channels)	<ul style="list-style-type: none">– Rising edge– Falling edge– Rising/falling edge		Read-to-clear <code>DPI_IRPTL_x</code> + RTI instruction	P14I

DAI/DPI Interrupt Mask Events

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.



Only the DAI interrupt controller latches interrupts on both edges. This ability does not exist in the core 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-214xx 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 three conditions:

- Latch on the rising edge
- Latch on the falling edge
- Latch on *both* the rising and falling edge

[Table 9-8](#) shows which interrupts are valid on rising and or falling edges

Table 9-8. DAI/DPI Interrupt Valid Edges

Interrupt Source	DAI_IMASK_RE	DAI_IMASK_FE
S/PDIF Rx	Yes	Yes
IDP_FIFO	Yes	No
IDP_DMA	Yes	No
SRC Mute Out	Yes	Yes
Miscellaneous (31–22)	Yes	Yes
	DPI_IMASK_RE	DPI_IMASK_FE
UART Channel Bits 3–0	Yes	No
TWI Bits (4)	Yes	No
Miscellaneous Interrupt Bits (5–13)	Yes	Yes

Use of the DAI_IMASK_RE or DAI_IMASK_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.

Interrupts

Enabling responses to changes in conditions of signals (including changes in DMA state, introduction of error conditions, and so on) can only be done using the `DAI_IMASK_RE` register.

DAI Interrupt Acknowledge

Any asynchronous or synchronous interrupt causes a latency, since it forces the core to stop processing an instruction in process, then vector to the interrupt service routine (ISR), (which is basically an interrupt vector table (IVT) lookup), then proceed to implement the instruction referenced in the IVT. For more information, see “[Peripheral Interrupt Control](#)” in [Appendix B, Peripheral Interrupt Control](#).

When an interrupt from the DAI must be serviced, one of the two core ISRs must query the DAI’s interrupt controller to determine the source(s). Sources can be any one or more of the interrupt controller’s 32 configurable channels (`DAI_INT31-0`). For more information, see “[Interrupt Controller Registers](#)” on page [A-149](#).



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

When a DAI interrupt occurs, the high or low priority core ISR queries its corresponding register to determine which of the 32 interrupt sources requires service. Sources can be any one or more of the interrupt controller’s 32 configurable channels. When `DAI_IMASK_H` is read, the high priority latched interrupts are cleared. When `DAI_IMASK_L` is read, the low priority latched interrupts are cleared.



Reading the DAI’s interrupt latches clears the interrupts (Read-to-Clear bit type). Therefore, the ISR must service *all* the interrupt sources it discovers. That is, if multiple interrupts are

latched in one of the `DAI_IMASK_x` registers, all of them must be serviced before executing an RTI instruction. [For more information, see “Interrupt Controller Registers” on page A-149.](#)

DPI Interrupt Acknowledge

Any asynchronous or synchronous interrupt causes a latency, since it forces the core to stop processing an instruction in process, then vector to the interrupt service routine (ISR), (which is basically an interrupt vector table (IVT) lookup), then proceed to implement the instruction referenced in the IVT.



The DPI triggers one interrupt in the primary IVT.

When a DPI interrupt occurs, the `DPI_IMASK` register determines which of the 12 interrupt sources requires service.



Reading the DPI’s interrupt latches (`DPI_IMASK`) clears the interrupts (Read-to-Clear bit type). Therefore, the ISR must service *all* the interrupt sources it discovers. That is, if multiple interrupts are latched in one of the registers, all of them must be serviced before executing an RTI instruction.

For UART and TWI interrupts in core operation, the interrupt acknowledge mechanisms may be different. For more information, refer to the specific chapters ([Chapter 20, UART Port Controller](#), [Chapter 21, Two Wire Interface Controller](#)).

Core versus DAI/DPI Interrupts

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 to which latch these interrupts are mapped.

Two registers (`DAI_IMASK_RE` and `DAI_MASK_FE`) replace the DAI peripheral’s version of the `IMASK` register. As with the `IMASK` register, these DAI

Debug Features

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.

-  The DAI/DPI interrupt controller has the same interrupt latency like the core interrupt controller which takes 6 cycle latency to respond to asynchronous interrupts.

Note that `IRPTL` and `LIRPTL` are system registers. All DAI interrupt registers (`DAI_IMASK_x`) are memory mapped registers.

Debug Features

The following sections describe features that can be used to help in debugging the DAI.

DAI Shadow Registers

The `DAI_IMASK_L_SH` and `DAI_IMASK_H_SH` shadow registers are provided for the `DAI_IMASK_L` and `DAI_IMASK_H` registers respectively. Reads of these registers returns data in `DAI_IMASK_L` and `DAI_IMASK_H` respectively without clearing the contents of these registers.

DPI Shadow Registers

The `DPI_IMASK_SH` shadow register is provided for register `DPI_IMASK`. A read of this register (RO) returns data in `DPI_IMASK` without clearing the contents of this register.

Loop Back Routing

The serial peripherals (SPORT and SPI) support an internal loop back mode. If the loop back bit for each peripheral is enabled, it connects the transmitter with the receiver block internally (does not signal off-chip).

The SRU can be used for this purpose. [Table 9-9](#) describes the different possible routings based on the peripheral.



The peripheral's loop back mode for debug is independent from both of the signal routing units.

Table 9-9. Loop Back Routing

Peripheral	Loopback Mode	SRU/SRU2 Internal Routing for Loopback	SRU/SRU2 External Routing for Loopback
DAI			
IDP	N/A	N/A	N/A
SPORT	Yes	SPORTx_xx_O → SPORTx_xx_I	SPORTx_xx_O → DAI_PBxx_I DAI_PBxx_O → SPORTx_xx_I
S/PDIF Tx/Rx	No	DIT_O → DIR_I	DIT_O → DAI_PBxx_I DAI_PBxx_O → DIR_I
SRC	No	SRCx_DAT_OP_O → SRCx_DAT_IP_I	SRCx_DAT_OP_O → DAI_PBxx_I DAI_PBxx_O → SRCx_DAT_IP_I
DPI			
Timer	No	TIMERx_O → TIMERx_I	TIMERx_O → DPI_PBxx_I DPI_PBxx_O → TIMERx_I
SPI	Yes	No	SPIx_xx_O → DPI_PBxx_I DPI_PBxx_O → SPIx_xx_I
UART0	No	UART0_TX_O → UART0_RX_I	UART0_TX_O → DPI_PBxx_I DPI_PBxx_O → UART0_RX_I
TWI	No	No	TWI_xx_O → DPI_PBxx_I DPI_PBxx_O → TWI_xx_I

Effect Latency

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

Write Effect Latency

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

Signal Routing Unit Effect Latency

After the DAI/DPI registers are configured the effect latency is 2 PCLK cycles minimum and 3 PCLK cycles maximum.

Programming Model

As discussed in the previous sections, the signal routing unit is controlled by writing values that correspond to signal sources into bit fields that further correspond to signal inputs. The SRU is arranged into functional groups such that the registers that are made up of these bit fields accept a common set of source signal values.

In order to ease the coding process, the header file `SRU.H` is included with the VisualDSP++ tools. This file implements a macro that abstracts away most of the work of signal assignments and functions. The macro has identical syntax in C/C++ and assembly, and makes a single connection from an output to an input as shown below.

```
SRU(Output Signal, Input Signal);
```

The names passed to the macro are the names given “[DAI Signal Routing Unit Registers](#)” on page A-118.

The following lines illustrate how the macro is used:

Listing 9-4. 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,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.

DAI Example System

A complete system using the DAI peripherals (SPORTs, PCG, S/PDIF) is shown in [Figure 9-16](#).

Programming Model

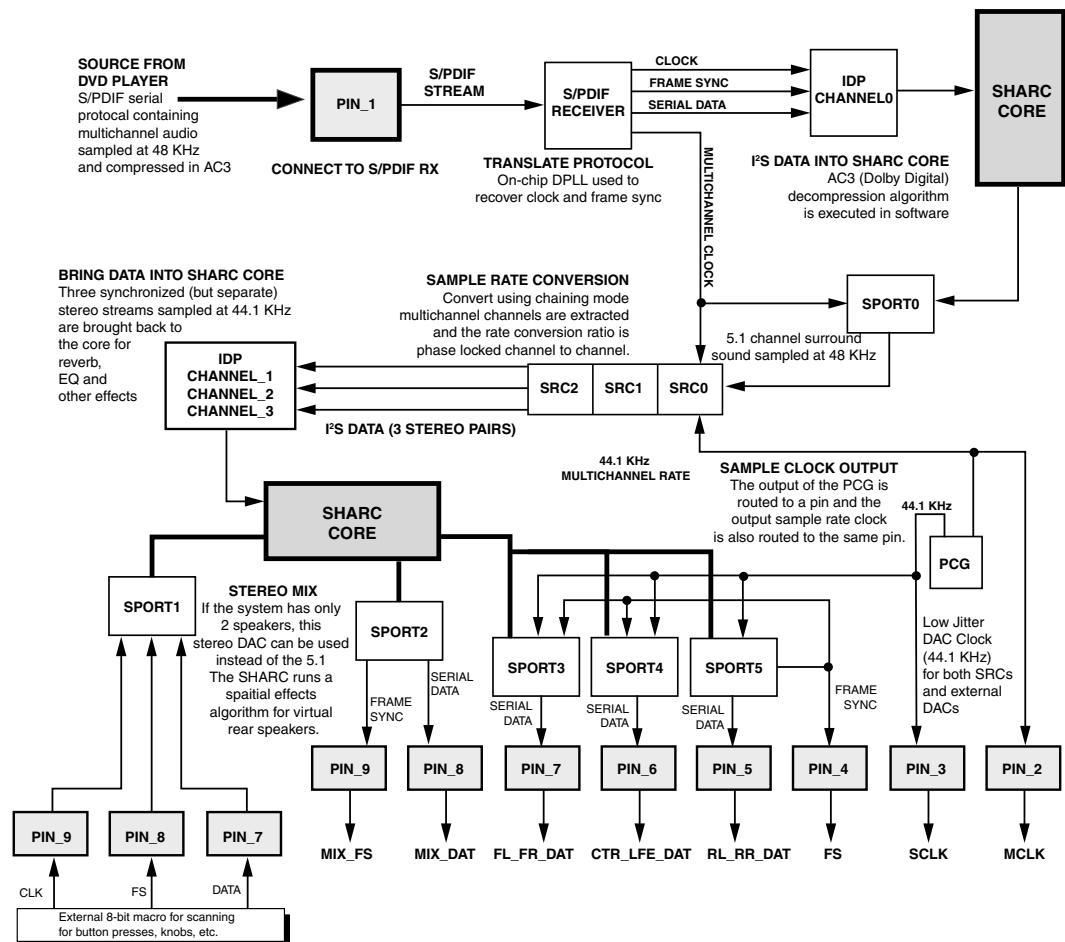


Figure 9-16. DAI Example

10 SERIAL PORTS

The processors have 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, SPORT5, 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. The interface specifications are shown in [Table 10-1](#).

Table 10-1. Serial Port Specifications

Feature	SPORT7–0[AB]
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	Yes
SRU DAI Default Routing	Yes
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	Yes
Transmission Half Duplex	Yes

Features

Table 10-1. Serial Port Specifications (Cont'd)

Feature	SPORT7-0[AB]
Transmission Full Duplex	No
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	2 per SPORT
DMA Chaining	Yes
Boot Capable	No
Local Memory	No
Clock Operation	$f_{PCLK}/4$ ($f_{PCLK}/8$, if SPORT is slave transmitter or master receiver)

Features

Serial ports offer the following features and capabilities:

- Five operation modes (“[Operation Modes](#)” on page 10-21):
 1. Standard serial
 2. Left-justified
 3. I²S
 4. Packed
 5. Multichannel
- Two bidirectional channels (A and B) per serial port, configurable as either transmitters or receivers. Each serial port can also be configured as two receivers or two transmitters, permitting two

unidirectional streams into or out of the same serial port. This bidirectional functionality provides greater flexibility for serial communications. Further, two SPORTs can be combined to enable full-duplex, dual-stream communications.

Serial ports can operate at a maximum of one-fourth the peripheral clock rate of the processor. If channels A and B are active, each SPORT has a maximum throughput of $2 \times \text{PCLK}/4$ rate.

- Chained DMA operations for multiple data blocks, see “[DMA Chaining](#)” on page 2-32.
- DMA Chain insertion mode allows the SPORTs to change DMA priority during chaining, see “[Enter DMA Chain Insertion Mode](#)” on page 10-56.
- 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 mode.
- 128-channel multichannel is supported in multichannel mode operation, useful for H.100/H.110 and other telephony interfaces described in “[Multichannel Mode](#)” on page 10-31.
- μ-law and A-law compression/decompression hardware companding on transmitted and received words when the SPORT operates in multichannel mode.



Receive comparison and 2-dimensional DMA are not supported.

Pin Descriptions

Table 10-2 describes pin function.

Table 10-2. SPORT Pin Descriptions

Internal Node	Direction	Description
SPORT7-0_DA_I/O	I/O	Data receive or transmit channel A. Bidirectional data pin. This signal can be configured as an output to transmit serial data, or as an input to receive serial data.
SPORT7-0_DB_I/O	I/O	Data receive or transmit channel B. Bidirectional data pin. This signal can be configured as an output to transmit serial data, or as an input to receive serial data.
SPORT7-0_CLK_I/O	I/O	Transmit/Receive Serial Clock. This signal can be either internally or externally generated.
SPORT7-0_FS_I/O	I/O	Transmit/Receive Frame Sync. The frame sync pulse initiates shifting of serial data. This signal is either generated internally or externally. It can be active high or low or an early or a late frame sync, in reference to the shifting of serial data.
SPORT7-0_TDV_O	O	Multichannel Transmit Data Valid. This signal is only active in SPORT multichannel mode configured as transmitter. The signal is asserted during enabled slots based on the transmit channel (companding) Selection registers (MTxCSSy/MTxCCSSy).
SPORT7-0_DA_PBEN_O	O	Only driven in master mode.
SPORT7-0_DB_PBEN_O	O	
SPORT7-0_CLK_PBEN_O	O	
SPORT7-0_FS_PBEN_O	O	
SPORT7-0_TDV_PBEN_O	O	

SRU Programming

Any of the serial port's signals can be mapped to digital applications interface (DAI_PX) pins through the signal routing unit (SRU) as shown in [Table 10-3](#). For more information, see [Chapter 9, Digital Application/Digital Peripheral Interfaces](#).

 SPORTs 6 and 7 receive their clocks from other routed sources but cannot route their own clocks to other SPORTs or other peripherals internally through the SRU. If SPORTs 6 and 7 are needed externally, they have to be routed through the DAI pins.

Table 10-3. SPORT DAI/SRU Signal Connections

Internal Node	DAI Connection	SRU Register
Inputs		
SPORT7-0_CLK_I	Group A	SRU_CLK1-0
SPORT7-0_FS_I	Group C	SRU_FSO
SPORT7-0_DA_I	Group B	SRU_DAT2-0
SPORT7-0_DB_I		
Outputs		
SPORT5-0_CLK_O	Group A, D	
SPORT7-6_CLK_O	Group D	
SPORT5-0_FS_O	Group C, D, E	
SPORT7-6_FS_O	Group D	
SPORT7-0_DA_O	Group B, D	
SPORT7-0_DB_O		
SPORT7-0_TDVO_O		

Table 10-3. SPORT DAI/SRU Signal Connections (Cont'd)

Internal Node	DAI Connection	SRU Register
SPORT7-0_CLK_PBEN_O	Group F	
SPORT7-0_FS_PBEN_O		
SPORT7-0_DA_PBEN_O		
SPORT7-0_DB_PBEN_O		
SPORT7-0_TDVB_PBEN_O		

SRU SPORT Receive Master

If the SPORT is operating as receive master, it must feed its master output clock back to its input clock. This is required to trigger the SPORT's state machine. Using SPORT 4 as an example receive master, programs should route `SPORT4_CLK_0` to `SPORT4_CLK_I`. This is not required if the SPORT is operating as a transmitter in master mode.

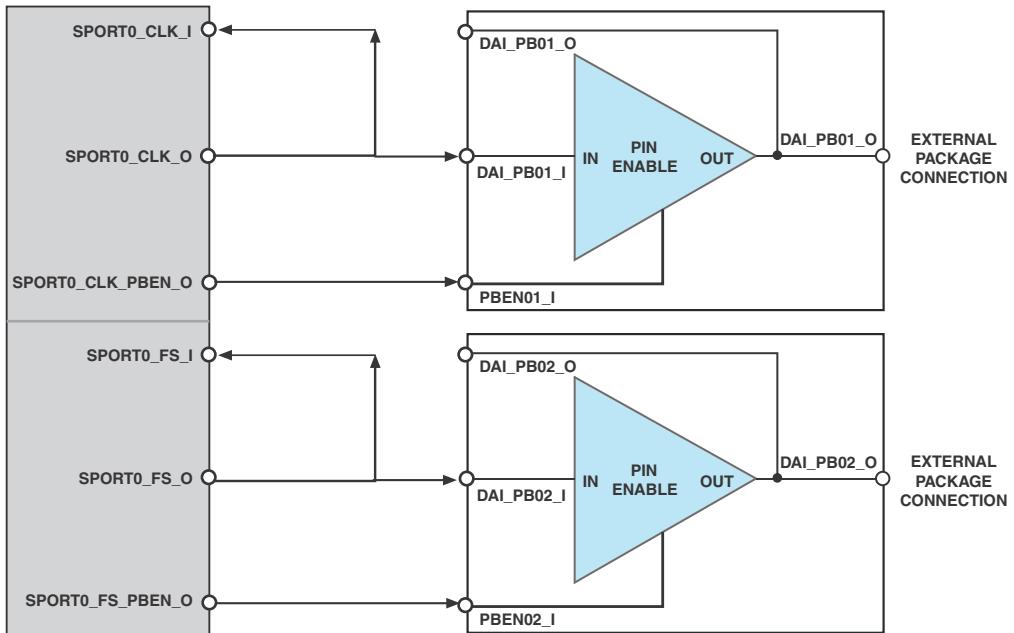
SRU SPORT Signal Integrity

There is some sensitivity to noise on the clock (`SPORTx_CLK`) and frame sync (`SPORTx_FS`) signals when the SPORT is configured as a master receiver. By correctly programming the signal routing unit (SRU) clock and frame sync registers, the reflection sensitivity in these signals can be avoided.

Figure 9-9 on page 9-19 shows the default routing of the serial port where the SRU maps to:

- The signal from the DAI pin (`DAI_PBxx_0`) back to the SPORT clock input (`SPORTx_CLK_I`)
- The SPORT clock output (`SPORTx_CLK_0`) to the pin buffer input (`DAI_PBxx_I`)

By redirecting the signals as shown in [Figure 10-1](#) where the clock and frame sync outputs are routed directly back to their respective inputs, the signal sensitivity issue can be avoided.



[Figure 10-1. SRU Configuration when SPORT is Master Receiver.](#)

Register Overview

This section provides brief descriptions of the major registers. For complete information, see [“Serial Port Registers” on page A-150](#).

Serial Port Control Registers (SPCTLx). The SPCTL_x registers control serial port modes and are part of the SPCTL_x (transmit and receive) control registers. Other bits in these registers set up DMA and I/O processor related serial port features. For information about configuring a specific

Clocking

operation mode, refer to [Table 10-7 on page 10-24](#) and “[Operation Modes](#)” on page 10-21.

Serial Port Error Registers (SPERRxx). Two error registers (SPERRCTLx/SPERRSTAT) are used to observe and control error handling during transfers. Detected errors can be frame sync violation or buffer over/underflow conditions. For more information, see “[Error Detection](#)” on page 10-51 and “[Error Status](#)” on page 10-53.

Multichannel Control Registers (SPMCTLx). There is one global control and status register for each SPORT (SPORT7–0) for multichannel operation. These registers define the number of channels, provide the status of the current channel, enable multichannel operation, and set the multichannel frame delay.

Master Clock Divider Registers (DIVx). The DIVx registers contain divisor values that determine frequencies for internally-generated clocks and frame syncs. If your system requires more precision and less noise and jitter, refer to [Chapter 14, Precision Clock Generator](#).

Clocking

The fundamental timing clock of the SPORT modules is peripheral clock/4 (PCLK/4). Each serial port has a clock signal (SPORTx_CLK) for transmitting and receiving data on the two associated data signals. The clock signals are configured by the ICLK and CKRE bits of the SPCTLx control registers. A single clock signal clocks both A and B data signals (either configured as inputs or outputs) to receive or transmit data at the same rate.

Master Clock

The CLKDIV bit field specifies how many times the processor’s internal clock (PCLK) is divided to generate the transmit and receive clocks. The

frame sync (`SPORTx_FS`) is considered a receive frame sync if the data signals are configured as receivers. Likewise, the frame sync `SPORTx_FS` is considered a transmit frame sync if the data signals are configured as transmitters. The divisor is a 15-bit value, (bit 0 in divisor register is reserved) allowing a wide range of serial clock rates. Use the following equation to calculate the serial clock frequency:

$$\text{Transmit master: } SCLK = PCLK \div (4(CLKDIV + 1))$$

$$\text{Receive master: } SCLK = PCLK \div (8(CLKDIV + 1))$$

The maximum serial clock frequency is equal to one-fourth (0.25) the processor's internal peripheral clock (`PCLK`) frequency, which occurs when `CLKDIV` is set to zero. Use the following equation to determine the value of `CLKDIV`, given the `PCLK` frequency and desired serial clock frequency:

$$CLKDIV = (PCLK \div 4 \times SCLK) - 1$$

If the serial clock of SPORT (`SCLK`) is required as general-purpose clock in a system, only the `ICLK/MSTR` bit and the serial clock divider register `DIVx` must be programmed.

Master Frame Sync

The bit field `FSDIV` specifies how many transmit or receive clock cycles are counted before a frame sync pulse is generated. In this way, a frame sync can initiate periodic transfers. The counting of serial clock cycles applies to internally- or externally-generated serial clocks. The formula for the number of cycles between frame sync pulses is:

$$\text{Number of serial clocks between frame syncs} = FSDIV + 1$$

Use the following equation to determine the value of `FSDIV`, given the serial clock frequency and desired frame sync frequency:

$$FSDIV = (SCLK \div FSCLK) - 1$$

Functional Description

The frame sync is continuously active when `FSDIV = 0`. The value of `FSDIV` should not be less than the serial word length minus one (the value of the `SLEN` field in the serial port control register), as this may cause an external device to abort the current operation or cause other unpredictable results. If the serial port is not being used, the `FSDIV` divisor can be used as a counter for dividing an external clock or for generating a periodic pulse or periodic interrupt. The serial port must be enabled for this mode of operation to work properly.



Programs should not use master clock/frame sync on SPORTs to drive ADCs/DACs in high fidelity audio systems. Use the precision clock generator (PCG) instead.

Slave Mode

Exercise caution when operating with externally-generated transmit clocks near the frequency of `PCLK/4` of the processor's internal clock. There is a delay between when the clock arrives at the `SPORTx_CLK` node and when data is output. This delay may limit the receiver's speed of operation. Refer to the appropriate product data sheet for exact timing specifications.

Externally-generated late transmit frame syncs also experience a delay from when they arrive to when data is output. This can also limit the maximum serial clock speed. Refer to the appropriate product data sheet for exact timing specifications.

Functional Description

The following sections provides general information on the function of the SPORTs.

- “[Architecture](#)” below

- “Data Types and Companding” on page 10-12
- “Frame Sync” on page 10-16

Architecture

A serial port receives serial data on one of its bidirectional serial data signals configured as inputs, or transmits serial data on the bidirectional serial data signals configured as outputs. It can receive or transmit on both channels simultaneously and unidirectionally, where the pair of data signals can both be configured as either transmitters or receivers.

The `SPORTx_DA` and `SPORTx_DB` channel data signals on each SPORT cannot transmit and receive data simultaneously for full-duplex operation.

Two SPORTs must be combined to achieve full-duplex operation. The `SPTRAN` bit in the `SPCTLx` register controls the direction for both the A and B channel signals.



The data direction of channel A and channel B on a particular SPORT must be the same.

Serial communications are synchronized to a clock signal. Every data bit must be accompanied by a clock pulse. Each serial port can generate or receive its own clock signal (`SPORTx_CLK`). Internally-generated serial clock frequencies are configured in the `DIVx` registers. The A and B channel data signals shift data based on the rate of `SPORTx_CLK`.

In addition to the serial clock signal, data may be signaled by a frame synchronization signal. The framing signal can occur at the beginning of an individual word or at the beginning of a block of words. The configuration of frame sync signals depends upon the type of serial device connected to the processor. Each serial port can generate or receive its own frame sync signal (`SPORTx_FS`) for transmitting or receiving data. Internally-generated frame sync frequencies are configured in the `DIVx` registers. Both the A and B channel data signals shift data based on their corresponding `SPORTx_FS` signal.

Functional Description

Figure 10-2 shows a block diagram of a serial port. Setting the SPTRAN bit enables the data buffer path, which, once activated, responds by shifting data in response to a frame sync at the rate of `SPORTx_CLK`. An application program must use the correct serial port data buffers, according to the value of `SPTRAN` bit. The `SPTRAN` bit enables either the transmit data buffers for the transmission of A and B channel data, or it enables the receive data buffers for the reception of A and B channel data. Inactive data buffers are not used.

The processor's SPORTs are not UARTs and cannot communicate with an RS-232 device or any other asynchronous communications protocol. One way to implement RS-232 compatible communication with the processor is to use two of the `FLAG` pins as asynchronous data receive and transmit signals.

Data Types and Companding

Linear transfers occur in the primary channel, if the channel is active and companding is not selected for that channel. Companded transfers occur if the channel is active and companding is selected for that channel. The multichannel compand select registers, `MTxCCSy` and `MRxCCSy`, specify the transmit and receive channels that are companded when multichannel mode is enabled.

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



The compression for transmission requires a minimum word length of 8 (`SLEN = 8`) for proper function. If `SLEN < 8` the expansion may not work correctly.

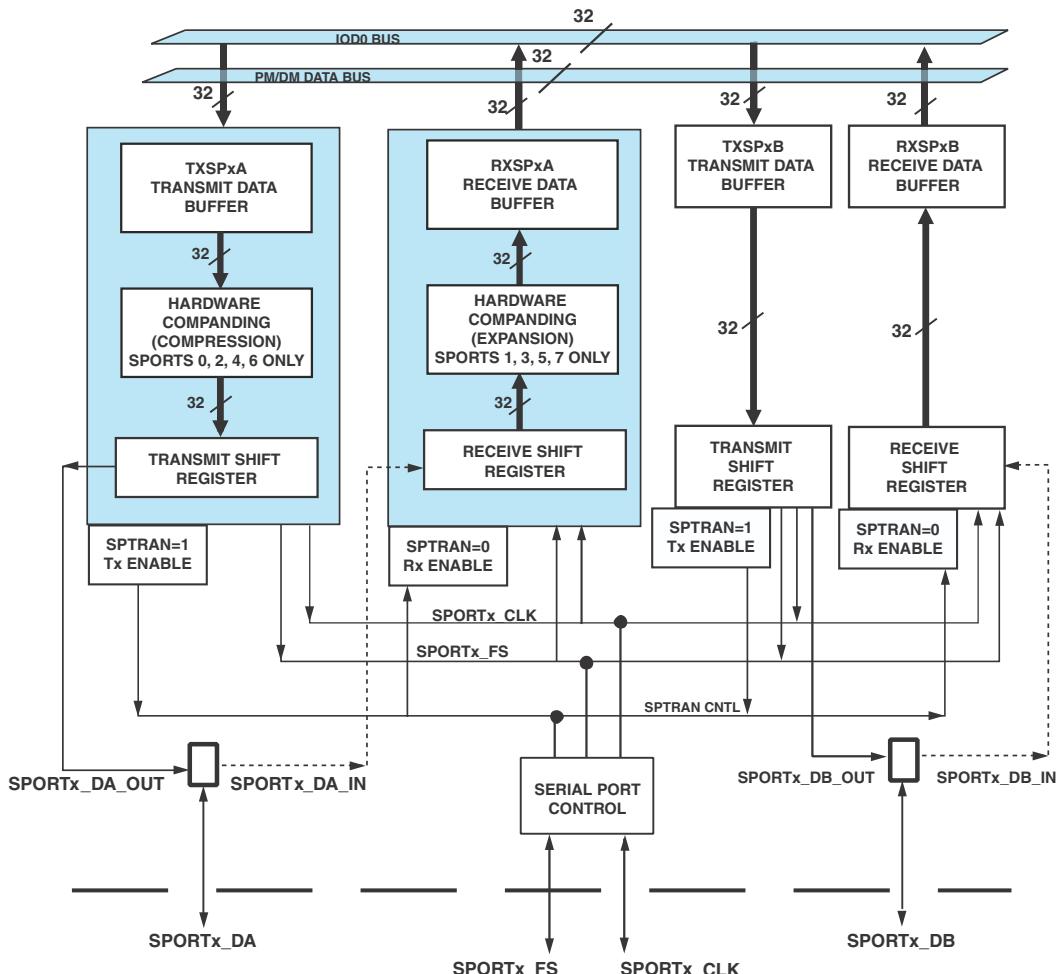


Figure 10-2. Serial Port Block Diagram

Companding the Data Stream

Companding (compressing/expanding) is the process of logarithmically encoding and decoding data to minimize the number of bits that must be sent. The processor's serial ports support the two most widely used

Functional Description

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 DTTYPE field of the SPCTLx control register.

-  Companding is supported on the A channel only. SPORT0, 2, 4 and 6 primary channels are capable of compression, while SPORTs 1, 3, 5 and 7 primary channels are capable of expansion.
In multichannel mode, when compression and expansion is enabled, the number of channels must be programmed via the NCH bit in the SPMCTLx registers before writing to the transmit buffer.
The SPxCSn and SPxCCSn registers should also be written before writing to transmit buffer.

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.

Transmit Path

If the serial port is configured as a serial transmitter, the data transmitted is written to the TXSPxA/TXSPxB buffer. The data is (optionally) compounded in hardware on the primary A channel (SPORT 0, 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 serial port is enabled as transmitter (`SPEN_A` or `SPEN_B` = 1 in the `SPCTLx` control register), unless it is in multichannel mode and an inactive time slot occurs.

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

When the SPORT is configured as a transmitter (`SPTRAN` = 1), the transmit buffers are activated. The transmit buffers act like a two-location FIFO because they have one data registers plus an output shift register.

Receive Path

If the serial data signal is configured as a serial receiver (`SPTRAN` = 0), the receive portion of the SPORT shifts in data from the `SPORTx_DA` or `SPORTx_DB` signal, synchronous to the `SPORTx_CLK` receive clock. If framing signals are used, the `SPORTx_FS` signal indicates the beginning of the serial word being received. When an entire word is shifted in on the primary A channel, the data is (optionally) expanded (SPORT1, 3, 5 and 7 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, and the channel receive shift registers respond 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.

Functional Description

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.

Frame Sync

The following sections provide information on frame syncs which applies to the SPORTs in all operating modes. For mode specific frame sync information, see “[Operation Modes](#)” on page 10-21.

Sampling Edge

Data and frame syncs can be sampled on the rising or falling edges of the serial port clock signals. The `CKRE` bit of the `SPCTLx` control registers selects the sampling edge. For sampling receive data and frame syncs, setting `CKRE` to 1 in the `SPCTLx` register selects the rising edge of `SPORTx_CLK`. When `CKRE` is cleared (=0), the processor selects the falling edge of `SPORTx_CLK` for sampling receive data and frame syncs.

Note that transmit data and frame sync signals change their state on the clock edge that is not selected. For example, the transmit and receive functions of any two serial ports connected together should always select the same value for `CKRE` so internally-generated signals are driven on one edge and received signals are sampled on the opposite edge.

Frame Sync and Data Sampling

The information contained in this section is generic to the SPORTs in any operating mode. Additional information about frame syncs and data sampling that applies to a specific operating mode can be found in “[Operation Modes](#)” on page 10-21.

As shown in [Figure 10-3](#) the SPORT uses two control signals to sample data.

1. Serial clock (**SCLK**) applies the bit clock for each serial data.
2. Frame sync (**FS**) divides the incoming data stream into frames.

Frames define the required data length (after the serial to parallel conversion) necessary to store the data in memory for further processing as shown in [Figure 10-3](#). The transmitter for example drives clock and frame sync called master while the receiver is slave sampling these data.

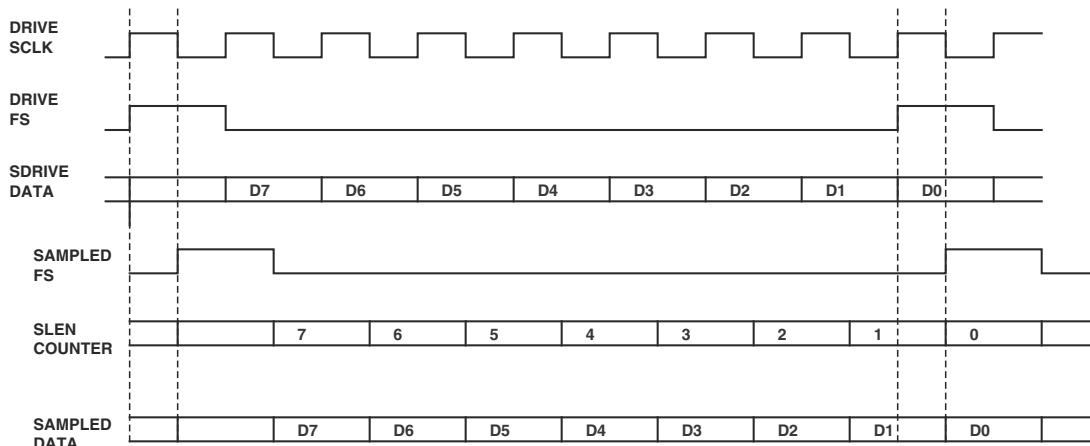


Figure 10-3. Frame Sync and Data Driven on Rising Edge

After the slave is sampled the **FS** the **SLEN** word counter is reloaded to the maximum setting. Each **SCLK** decrements the **SLEN** counter until the full frame is received. If the transmitter drives the frame sync and data on the rising edge, the falling edge is used to sample the frame sync and data, and vice versa.

Functional Description

Serial Word Length

The serial word length is not unique and is based on the operation mode. Moreover the companding feature limits the word length settings.

Words smaller than 32 bits are right-justified in the receive and transmit buffers, residing in the least significant (LSB) bit positions ([Table 10-4](#)).

Table 10-4. Data Length Versus Modes

Mode	Word Length (SLEN) Bits
Standard Serial Mode	3–32
Left justified	8–32
I ² S	8–32
Packed	3–32
Multichannel	3–32

Internal Versus External Frame Syncs

Both transmit and receive frame syncs can be generated internally or input from an external source. The IFS/IMFS bit of the SPCTL_X control register determines the frame sync source.

When IFS/IMFS is set (=1), the corresponding frame sync signal is generated internally by the processor, and the SPORT_X_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 DIV_X register.

When IFS/IMFS is cleared (=0), the corresponding frame sync signal is accepted as an input on the SPORT_X_FS signals, and the frame sync divisors in the DIV_X registers are ignored.

All frame sync options are available whether the signal is generated internally or externally.

Note that for I²S and left-justified mode, the MSTR bit allows programs to select only the clock and frame sync to be simultaneously configured as master or slave.

External Frame Sync Sampling

A variety of framing options are available on the SPORTs as shown in [Table 10-5](#).



When the SPORT is enabled, an already active externally applied frame sync is not allowed to start operation. An additional feature allows programs to configure the SPORTs to wait for a valid state change from inactive to active for the external frame sync to consider it valid. This is true for the first valid frame after the SPORT is enabled and applicable to both level and edge sensitive frame syncs. This feature is enabled by setting the frame sync edge detection bit (FSED, bit 2 in the SPCTLNx register).

Table 10-5. Framing Options

OPMODE	External Frame Sync Sampling
Standard serial	Level sensitive
Left-justified pair	Edge detection
I ² S	Edge detection
Packed	Edge detection
Multichannel	Level/edge sensitive

Signals: Level Versus Edge Sampling

The SPORT slaves allow programs to sample the frame sync and data on its signal level or edges depending on the operation mode. Note that in a noise free environment it doesn't matter which sampling type is selected.

In noisy environments however, the edge based sampling is preferred as it allows better re-synchronization of the communication link.

Functional Description

Logic Level Frame Syncs

Frame sync signals may be active high or active low (for example, inverted). The LFS/LMFS bit in the SPCTLx registers selects the logic level of the frame sync signals as active low (inverted) if set (=1) or active high if cleared (=0). Active high (=0) is the default.

Data-Independent Frame Sync

When DIFS = 0 and SPTRAN = 1, the internally-generated transmit frame sync is only output when a new data word has been loaded into the SPORT channel's transmit buffer. Once data is loaded into the transmit buffer, it is not transmitted until the next frame sync is generated. This mode of operation allows data to be transmitted only at specific times. When DIFS = 0 and SPTRAN = 0, a receive SPORTx_FS signal is generated only when receive data buffer status is not full.

The data-independent frame sync mode allows the continuous generation of the SPORTx_FS signal, with or without new data in the buffers. The DIFS bit of the SPCTLx control register configures this option. When DIFS = 1 and SPTRAN = 1, a transmit SPORTx_FS signal is generated regardless of the transmit data buffer status. When DIFS = 1 and SPTRAN = 0, a receive SPORTx_FS signal is generated regardless of the receive data buffer status.

Note that the SPORT DMA controller typically keeps the transmit buffer full. The application is responsible for filling the transmit buffers with data.

Operation Modes

SPORTs operate in five modes:

- Standard serial mode, described in “[Standard Serial Mode](#)” on [page 10-25](#)
- Left-justified mode, described in “[Left-Justified Mode](#)” on [page 10-28](#)
- I²S mode, described in “[I²S Mode](#)” on [page 10-30](#)
- Packed mode, described in “[Packed Mode](#)” on [page 10-37](#)
- Multichannel mode, described in “[Multichannel Mode](#)” on [page 10-31](#)



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



Pairings of SPORTs (0 and 1, 2 and 3, 4 and 5 and 6 and 7) are only used in loopback mode for testing.

The main control register for each serial port is the serial port control register, SPCTLx. These registers are described in “[Serial Port Registers](#)” on [page A-150](#).



When changing operating modes, clear the serial port control register before the new mode is written to the register.

The SPCTLx registers control the operating modes of the serial ports. [Table 10-6](#) lists all the bits in the SPCTLx register.

Operation Modes

Table 10-6. SPCTLx Control Bit Comparison

Bit	Standard Serial Mode	I ² S and Left-justified Mode	Packed Mode	Multichannel Mode
Control				
0		SPEN_A		Reserved
1–2	DTYPE	Reserved		DTYPE
3	LSBF	Reserved		LSBF
4–8			SLEN	
9			PACK	
10	ICLK	MSTR		ICLK
11			OPMODE	
12	CKRE	Reserved		CKRE
13	FSR		Reserved	
14	IFS	Reserved		IMFS
15		DIFS		Reserved
16	LFS		L_FIRST	LMFS
17	LAFS	OPMODE		Reserved
18			SDEN_A	
19			SCHEN_A	
20			SDEN_B	
21			SCHEN_B	
22	FS_BOTH		Reserved	
23			BHD	
24		SPEN_B		Reserved
25			SPTRAN	

Table 10-6. SPCTLx Control Bit Comparison (Cont'd)

Bit	Standard Serial Mode	I ² S and Left-justified Mode	Packed Mode	Multichannel Mode
Status				
26		DERR_B		
27–28		DXS_B		
29		DERR_A		
30–31		DXS_A		

Mode Selection

The serial port operating mode can be selected via the SPCTLx and the SPMCTLx/y registers.

1. The operating mode bit 11 (OPMODE) of the SPCTLx register selects between I²S, left-justified, and standard serial/multichannel mode.
2. The operating mode bit 17 (OPMODE) of the SPCTLx register selects between I²S mode and left-justified mode.
3. For packed mode, bit 11 (OPMODE) of the SPCTLx register and bit 0 (MCEA) in the SPMCTLx register enables the A channels and bit 23 (MCEB) in the SPMCTLx register enables the B channels.
4. In multichannel mode, the bit 0 (MCEA) in the SPMCTLx register enables the A channels and the bit 23 (MCEB) in the SPMCTLx register enables the B channels.
5. The OPMODE bit 17 serves for standard serial mode as late frame sync bit (LAFS).

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 10-7 provides values for each of the bits in the SPORT serial

Operation Modes

control (SPCTLx) registers that must be set in order to configure each specific SPORT operation mode. The shaded columns indicate that the bits come from different control registers.

Table 10-7. SPORT Operation Modes

OPERATING MODES (x = A or B or A and B SPORT Channels)	SPCTLx Bits			SPMCTLx Bits
	OPMODE (Bit 11)	OPMODE (Bit 17)	SPEN_x (Bit 0/24)	
Standard Serial Mode	0	Valid	1	0
Left-justified Mode	1	1	1	0
I ² S Mode	1	0	1	0
Packed Mode	1	0	0	1
Multichannel Mode	0	0	0	1

The following sections provide detailed information on each operating mode available using the serial ports. It should be noted that many bits in the SPORT registers that control the function of the mode are the same bit but have a different name depending on the operating mode. Further, some bits are used in some modes but not others. For reference, see [Table 10-6 on page 10-22](#), [Table 10-7](#), and “Serial Port Registers” on [page A-150](#).

Channel Order First

For left-justified, I²S and packed modes the next table demonstrates which word is transmitted or receive first depending on the L_FIRST bit.

Table 10-8. Channel Order First

OPMODE	L_FIRST = 0	L_FIRST = 1
Left-Justified	Data first after rising edge	Data first after falling edge
Packed	Data first after rising edge	Data first after falling edge
I ² S	Data first after falling edge	Data first after rising edge

Standard Serial Mode

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

Timing Control Bits

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

- Internal Clock (ICLK)
- Internal Frame Sync (IFS)
- Frame Sync Required (FSR)
- Sampling Edges Frame Sync/data (CKRE)
- Logic Level Frame Sync (LFS)
- Late Frame Sync (LAFS)
- Word length (SLEN, 3–32 bits)
- Word Order (LSBF)
- Word Packing (PACK)

Operation Modes

Clocking Options

In standard serial mode, the serial ports can either accept an external serial clock or generate it internally. The `ICLK` bit in the `SPCTL` register determines the selection of these options. For internally-generated serial clocks, the `CLKDIV` bits in the `DIVx` register configure the serial clock rate.

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.

Frame Sync Options

This section describes the different options for frame sync in standard serial mode.

Framed Versus Unframed Frame Syncs

The use of frame sync signals is optional in serial port communications. The `FSR` (transmit frame sync required) bit determines whether frame sync signals are required. Active low or active high frame syncs are selected using the `LFS` bit. This bit is located in the `SPCTLx` control registers.

When `FSR` is set (=1), a frame sync signal is required for every data word. To allow continuous transmission from the processor, each new data word must be loaded into the transmit buffer before the previous word is shifted out and transmitted.

When `FSR` is cleared (=0), the corresponding frame sync signal is not required. A single frame sync is required to initiate communications but it is ignored after the first bit is transferred. Data words are then transferred continuously in what is referred to as an unframed mode.



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 10-4 illustrates framed serial transfers.

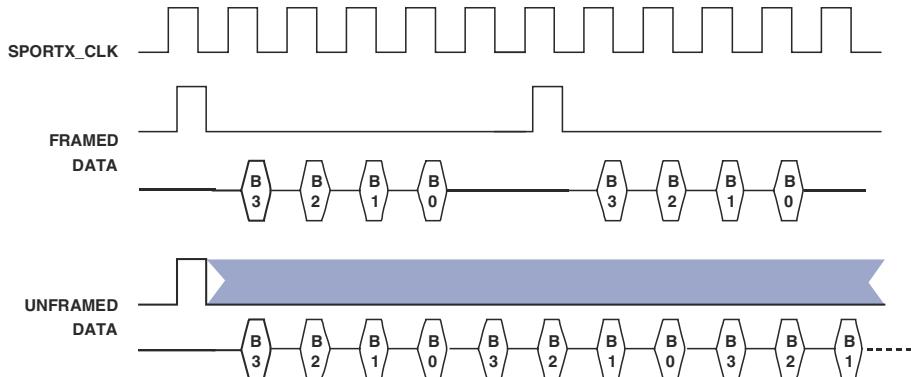


Figure 10-4. Framed Versus Unframed Data

Early Versus Late Frame Syncs

Frame sync signals can be early or late. Frame sync signals can occur during the first bit of each data word or during the serial clock cycle immediately preceding the first bit. The `LAFS` bit of the `SPCTLx` control register configures this option.

When `LAFS` is cleared ($=0$), early frame syncs are configured. This is the normal mode of operation. In this mode, the first bit of the transmit data word is available (and the first bit of the receive data word is latched) in the serial clock cycle after the frame sync is asserted. The frame sync is not checked again until the entire word has been transmitted (or received). In multichannel operation, this is the case when the frame delay is one.

If data transmission is continuous in early framing mode (for example, the last bit of each word is immediately followed by the first bit of the next word), the frame sync signal occurs during the last bit of each word. Internally-generated frame syncs are asserted for one clock cycle in early framing mode.

Operation Modes

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 10-5 illustrates the two modes of frame signal timing.

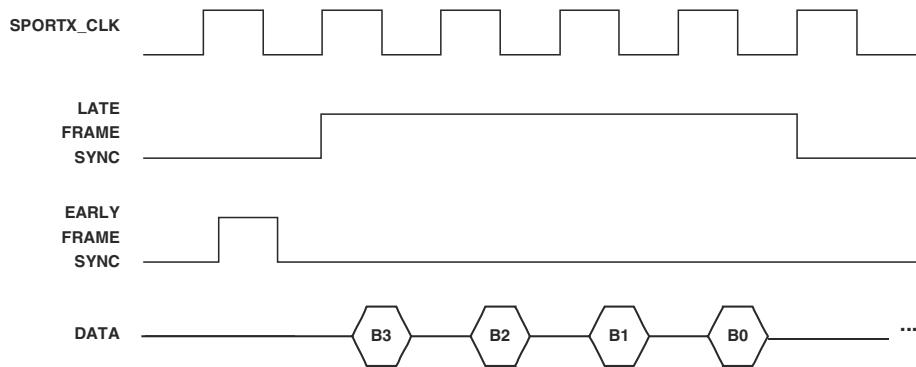


Figure 10-5. Normal Versus Alternate Framing

Left-Justified Mode

Left-justified mode is a mode where in each frame sync cycle two samples of data are transmitted/received—one sample on the high segment of the frame sync, the other on the low segment of the frame sync.



Companding is not supported in left-justified or I²S mode.

Master Serial Clock and Frame Sync Rates

The serial clock rate (`CLKDIV` value) for internal clocks can be set using a 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 based on the `MSTR` bit setting.

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

To transmit or receive words continuously in left-justified mode, load the `FSDIV` register with `SLEN`–1. For example, for 8-bit data words set `FSDIV` = 7.

Timing Control Bits

Several bits in the `SPCTLx` control register enable and configure left-justified mode operation:

- Master Mode Clock and Frame Sync (`MSTR`)
- Word Length (`SLEN`, 8–32 bits)
- Channel Order (`L_FIRST`)
- Word Packing (`PACK`)

[Figure 10-6](#) illustrates only one possible combination of settings attainable in the left-justified mode. In this example case, `OPMODE` = 1, `LAFS` = 1, and `L_FIRST` = 0. For complete descriptions of these bits, see “[Serial Control Registers \(SPCTLx\)](#)” on page [A-151](#).

Operation Modes

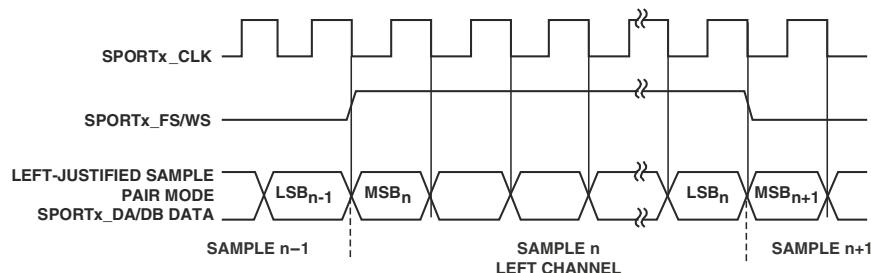


Figure 10-6. Word Select Timing in Left-Justified Mode

I²S Mode

The following sections provide information on using I²S mode.

Master Serial Clock and Frame Sync Rates

The serial clock rate (`CLKDIV` value) for internal clocks can be set using a 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 based on the `MSTR` bit setting.

The transmitter sends the MSB of the next word in the same clock cycle as the word select (`SPORTx_FS`) signal changes. To transmit or receive words continuously in I²S mode, load the `FSDIV` register with `SLEN-1`. For example, for 8-bit data words set `FSDIV = 7`.

Timing Control Bits

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

- Master Mode Clock and Frame Sync (`MSTR`)
- Sampling Edges Frame Sync/Data (`CKRE`)

- Word length ($SLEN$, 8–32 bits)
- Channel Order (L_FIRST)
- Word Packing ($PACK$)

i I²S mode is simply a subset of the left-justified mode. Note that in I²S mode, the data is delayed by one $SCLK$ cycle and the operation transfer starts on the left channel first ($L_FIRST = 1$).

i When both SPORT channels A and B are used in I²S/left-justified mode with standard DMA enabled, then the DMA count should be the same for both channels.

Figure 10-7 illustrates timing in I²S mode. In this example case, $OPMODE = 1$, $LAFS = 1$, and $L_FIRST = 0$.

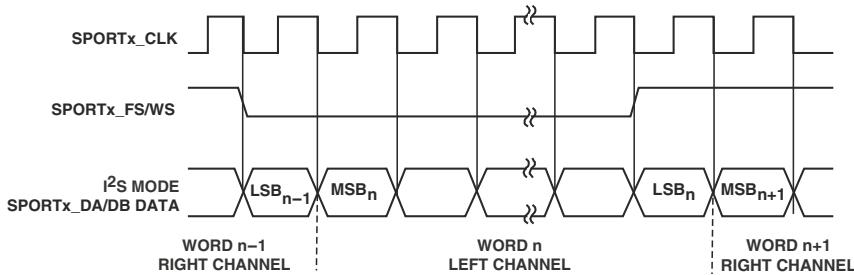


Figure 10-7. Word Select Timing in I²S Mode

Multichannel Mode

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

Operation Modes

The serial port can automatically select some words for particular channels while ignoring others. Up to 128 channels are available for transmitting or receiving or both. Each SPORT can receive or transmit data selectively from any of the 128 channels.

Data companding and DMA transfers can also be used in multichannel mode on channel A. Channel B can also be used in multichannel mode, but companding is not available on this channel.

Although the eight SPORTs are programmable for data direction in the standard mode of operation, their programmability is restricted for multi-channel 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 compression only.
2. Receive comparison is not supported.

Clocking Options

In multichannel mode, the serial ports can either accept an external serial clock or generate it internally. The `ICLK` bit in the `SPCTL` register determines the selection of these options. For internally-generated serial clocks, the `CLKDIV` bits in the `DIVx` register configure the serial clock rate. Finally, programs can select whether the serial clock rising 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.

Frame Sync Options

In previous SHARC processors, multichannel mode required a SPORT pair. This pair was needed to route the `SCLK` on the even SPORT and the frame sync to the odd SPORT. The pair itself interconnect the `SCLK` and `FS` signals.



Multichannel mode operates completely independently and each uses its own SCLK and FS signal programmed using the SRU. The FS signal synchronizes the channels and restarts each multichannel sequence. The `SPORTx_FS` signal initiates the start of the channel 0 data word. The FS period in multichannel is defined as:

$\text{FS period} = \text{SLEN} \times \text{number of channels}$. The frame sync can be configured in master or slave mode based on the setting of the `IMFS` bit and the logic level can be changed using the `LMFS` bit. The edge can be changed if bit 2 (`FSED`) is set in the `SPCTLNx` register.

Frame Sync Delay (MFD)

The 4-bit MFD field (bits 4–1) in the multichannel control registers (`SPMCTLx`) specifies a delay between the frame sync pulse and the first data bit in multichannel mode. The value of `MFD` is the number of serial clock cycles of the delay. Multichannel frame delay allows the processor to work with different types of telephony interface devices.

A value of zero for `MFD` causes the frame sync to be concurrent with the first data bit. The maximum value allowed for `MFD` is 15. A new frame sync may occur before data from the last frame has been received, because blocks of data occur back to back.

Transmit Data Valid Signal

Each SPORT has its own transmit data valid signal (`SPORTx_TDVO`) which is active during the transmission of an enabled word. Because the serial port's receiver signals are three-stated when the time slot is not active, the `SPORTx_TDVO` signal specifies if the SPORT data is being driven by the processor.

After the `TXSPxA` transmit buffer is loaded, transmission begins and the `SPORTx_TDVO` signal is asserted.

Operation Modes

Figure 10-8 shows an example of timing for a multichannel transfer with SPORT pairing using SPORT0 and 1. The transfer has the following characteristics.

- SPORT1–0 have the same SCLK and frame sync as input.
- Multichannel is configured as 8 channels.
- SPORT0A drives data to DAC1 during slot 1–0 which asserts TDV for 2 slots.
- SPORT1A drives data to DAC2 during slot 3–2 which asserts TDV for 2 slots.
- SPORT1B receives data from ADC during slot 3–0.

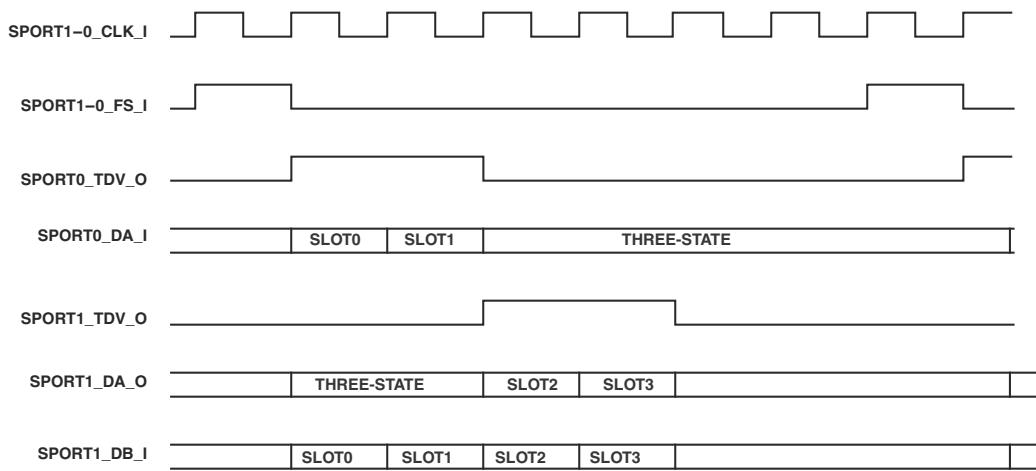


Figure 10-8. Multichannel Operation

Transmit Data Valid Output

In the ADSP-214xx processors, each SPORT has its own transmit data valid output signal (SPORTx_TDVO) which is active during the transmission of an enabled word. Because the serial port's receiver signals are

three-stated when the time slot is not active, the `SPORTx_TDV_0` signal specifies if the SPORT data is being driven by the processor.

Unlike previous SHARC processors, the assertion of the `SPORTx_TDV_0` is independent for the transmit buffer status (valid data or not). So writing to the buffer does not affect the `SPORTx_TDV_0` output timing.

Timing Control Bits

Several bits in the `SPCTLX` register enable and configure multichannel mode.

- Frame Delay (`MFD`)
- Number of multichannel channels (`NCH`)
- Internal Clock (`ICLK`)
- Internal Frame Sync (`IMFS`)
- Sampling Edges Frame Sync/Data (`CKRE`)
- Logic Level Frame Sync (`LMFS`)
- Word Length (`SLEN`, 8–32 bits)
- Word Order (`LSBF`)
- Word Packing (`PACK`)

Number of Channels (`NCH`)

Select the number of channels used in multichannel operation by using the 7-bit `NCH` field in the multichannel control register. Set `NCH` to the actual number of channels minus one (`NCH` = Number of channels – 1).

The 7-bit `CHNL` field in the multichannel control registers indicates the channel that is currently selected during multichannel operation. This

Operation Modes

field is a read-only status indicator. The CHNL(6:0) bits increment modulo NCH(6:0) as each channel is serviced.

Active Channel Selection Registers

Specific channels can be individually enabled or disabled to select the words that are received and transmitted during multichannel communications. Data words from the enabled channels are received or transmitted, while disabled channel words are ignored. Up to 128 channels are available for transmitting and receiving.

The multichannel selection registers enable and disable individual channels. The registers for each serial port are shown in “[Serial Port Registers](#)” on page A-150.

Each of the four multichannel enable and compand select registers are 32 bits in length. These registers provide channel selection for 128 (32 bits \times 4 channels = 128) channels. Setting a bit enables that channel so that the serial port selects its word from the multiple-word block of data (for either receive or transmit). For example, setting bit 0 for TX SPORT0 and TX SPORT7 (MTOCS0 or MT7CS0) selects channel 0, setting bit 12 selects channel 12, and so on. Setting bit 0 for TX SPORT0 and TX SPORT7 (MTOCS1 or MT7CS1) selects channel 32, setting bit 12 selects channel 44, and so on.

Companding Selection

Companding may be selected on a per-channel basis. Setting a bit to 1 in any of the multichannel registers (MTxCCSy or MRxCCSy) specifies that the data be companded for that channel. A-law or μ -law companding can be selected using the DTYPE bit in the SPCTLx control registers. SPORTA1, 3, 5 and 7 expand selected incoming time slot data, while SPORTA0, 2, 4 and 6 can compress the data.

Companding Limitations (ADSP-2146x)

In multichannel mode there is an option to enable companding for any active channel. If the first active channel is NOT the channel 0 and companding is enabled for the first active channel (channel 2), then from the second frame onward companding for the first active channel (channel 2) does not occur. In [Table 10-9](#), x = Don't care 0 = Not Active 1 = Active.

Table 10-9. Companding

Channel Number	0	1	2	3	4	5
Active Channel Number	0	0	1	x	x	x
Companding Enable	0	0	1	x	x	x

In [Table 10-9](#) channel 0 and 1 are not active and channel 2 is active and companding is enabled. In the first frame companding occurs for the first active channel (channel 2) but the second frame onward companding for the first active channel (channel 2) does not occur. However, for other channels, companding occurs correctly.

Packed Mode

A packed mode is available in the SPORT and used for audio codec communications using multiples channels. This mode allows applications to send more than the standard 32 bits per channel available through standard I²S mode. Packed mode is implemented using standard multichannel mode (and is therefore programmed similarly to multichannel mode). Packed mode also supports the maximum of 128 channels as does multichannel mode as well as the maximum of (128 x 32) bits per left or right channel.

As shown in [Figure 10-9](#), packed waveforms are the same as the waveforms used in multichannel mode, except that the frame sync is toggled

Operation Modes

for every frame, and therefore emulates I²S mode. So it is a hybrid between multichannel and I²S mode.

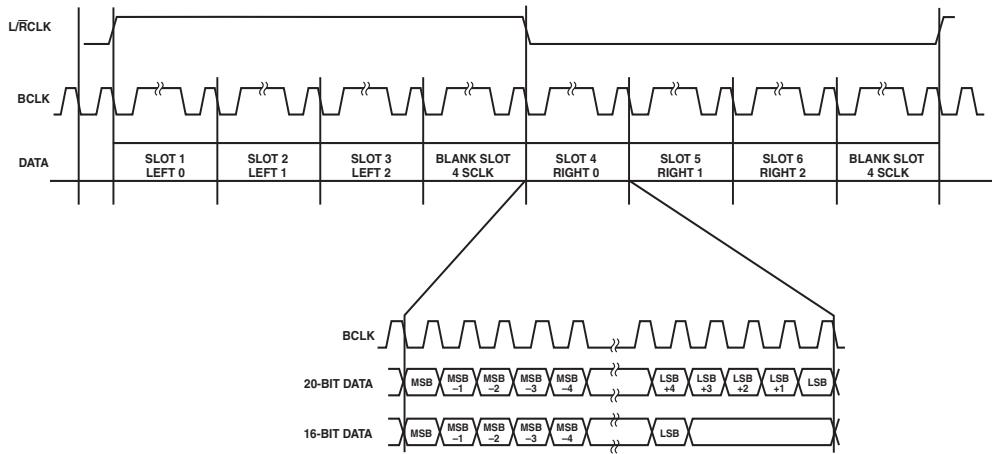


Figure 10-9. Packed Mode 128 Operation

Clocking Options

In packed mode, the serial ports can either accept an external serial clock or generate it internally. The `ICLK` bit in the `SPCTL` register determines the selection of these options. For internally-generated serial clocks, the `CLK-DIVx` bits in the `DIVx` register configure the serial clock rate. 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.

Frame Sync Options

The frame sync period in packed mode is defined as:

$$\text{FS period} = \text{SLEN} \times \text{number of channels.}$$

The frame sync can be configured in master or slave mode depending on the `IMFS` bit. Moreover the logic level can be changed with the `LMFS` bit.

Timing Control Bits

Several bits in the SPCTL_x register enable and configure packed mode.

- Internal Clock (ICLK)
- Internal Frame Sync (IFS)
- Sampling Edges Frame Sync/Data (CKRE)
- Selecting Channel Order (L_FIRST)
- Word Length (SLEN, 8–32 bits)
- Word Order (LSBF)
- Word Packing (PACK)

The following bits in the SPMCTL_x register are used to configure timing options in packed mode.

- Frame Delay (MFD)
- Number of multichannel channels (NCH)

Data Transfers

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

- Core-driven single word transfers
- DMA transfers between both internal and external memory

DMA transfers can be set up to transfer a configurable number of serial words between the serial port buffers (TXSPxA, TXSPxB, RXSPxA, and RXSPxB) and internal memory automatically. Core-driven transfers use SPORT interrupts to signal the processor core to perform single word

Data Transfers

transfers to/from the serial port buffers (TXSPxA, TXSPxB, RXSPxA, and RXSPxB).

Data Buffers

When programming the serial port channel (A or B) as a transmitter, only the corresponding TXSPxA and TXSPxB buffers become active while the receive buffers RXSPxA and RXSPxB remain inactive. Similarly, when the SPORT channel A and B are programmed as receive-only the corresponding RXSPxA and RXSPxB are activated. Do not attempt to read or write to inactive data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, unpredictable results may occur.



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

Transmit Buffers (TXSPxA/B)

The transmit buffers (TXSP7-0A, TXSP7-0B) are the 32-bit transmit data buffers for SPORT7-0 respectively. These buffers must be loaded with the data to be transmitted if the SPORT is configured to transmit on the A and B channels. The data is loaded automatically by the DMA controller or loaded manually by the program running on the processor core.

The transmit buffers act like a two-location buffer because they have a data register plus an output shift register. Two 32-bit words may both be stored in the transmit queue at any one time. When the transmit register is loaded and any previous word has been transmitted, the register contents are automatically loaded into the output shifter. An interrupt occurs when the output transmit shifter has been loaded, signifying that the transmit buffer is ready to accept the next word (for example, the transmit buffer is not full). This interrupt does not occur when serial port DMA is enabled or when the corresponding mask bit in the LIRPTL/IRPTL register is set.

Receive Buffers (RXSPxA/B)

The receive buffers (RXSP7-0A, RXSP7-0B) are the 32-bit receive data buffers SPORT7-0 respectively. These 32-bit buffers become active when the SPORT is configured to receive data on the A and B channels. When a SPORT is configured as a receiver, the RXSPxA and RXSPxB registers are automatically loaded from the receive shifter when a complete word has been received. The data is then loaded to internal memory by the DMA controller or read directly by the program running on the processor core.

Buffer Status

Serial ports provide status information about data buffers via the DXS_A and DXS_B status bits and error status via DERR_x bits in the SPCTL register. Depending on the SPTRAN setting, these bits reflect the status of either the TXSPxy or RXSPxy data buffers.

If your program causes the core processor to attempt to read from an empty receive buffer or to write to a full transmit buffer, the access is delayed until the buffer is accessed by the external I/O device. This delay is called a core processor hang. If you do not know if the core processor can access the receive or transmit buffer without a hang, the buffer's status should be read first (in SPCTLx) to determine if the access can be made.

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

Two complete 32-bit words can be stored in the receive buffer while a third word is being shifted in. The third word overwrites the second if the first word has not been read out (by the processor core or the DMA controller). When this happens, the receive overflow status bit is set in the serial port control register. Almost three complete words can be received without the receive buffer being read before an overflow occurs. The overflow status is generated on the last bit of the third word. The DERR_x status bits are sticky and are cleared only by disabling the serial port.

Data Transfers

 If the SPORTs are configured as transmitters, programs should not write to the inactive TXSPxA and TXSPxB buffers. If the core keeps writing to the inactive buffer, the transmit buffer status becomes full. This causes the core to hang indefinitely since data is never transmitted to the output shift register.

If the SPORTs are configured as receivers, programs should not read from the inactive RXSPxA and RXSPxB buffers. If the core keeps reading from to the inactive buffer, the receive buffer status becomes empty. This causes the core to hang indefinitely since new data is never received via the input shift register.

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

Data Buffer Packing

Received data words of 16 bits or less may be packed into 32-bit words, and 32-bit words being transmitted may be unpacked into 16-bit words. Word packing and unpacking is selected by the PACK bit in the SPCTLx control registers.

When `PACK = 1` in the control register, two successive words received are packed into a single 32-bit word, and each 32-bit word is unpacked and transmitted as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15–0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31–16. This applies to both receive (packing) and transmit (unpacking) operations. Companding can be used with word packing or unpacking.

When serial port data packing is enabled, the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.



When 16-bit received data is packed into 32-bit words and stored in normal word space in processor internal memory, the 16-bit words can be read or written with short word space addresses.

Core Transfers

The following sections provide information on core driven data transfers.

Single Word Transfers

Individual data words may also be transmitted and received by the serial ports, with interrupts occurring as each 32-bit word is transmitted or received. When a serial port is enabled and DMA is disabled, the SPORT interrupts are generated whenever a complete 32-bit word has been received in the receive buffer, or whenever the transmit buffer is not full.

When performing core-driven transfers, write to the buffer designated by the SPTRAN bit setting in the SPCTLX register. For DMA-driven transfers, the serial port logic performs the data transfer from internal memory to/from the appropriate buffer depending on the SPTRAN bit setting. If the inactive SPORT data buffers are read or written to by core while the port is being enabled, the core 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.

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

Data Transfers

Frame Sync Generation

The frame syncs are generated if the transmit or receive buffers are updated according to the `DIFS` bit setting (=0). If there is no buffer update by the core, the frame sync out is not driven off-chip and data output is zero.

If both A and B channels are enabled, one of the following can occur.

- In standard mode the `FS_BOTH` bit (in the `SPCTLX` register) defines the conditions of whether both channels are logically ANDed or ORed.
- For all other operating modes, channels A and B are logically ANDed. If both channels are enabled, both buffers need to be updated to drive data and frame sync off-chip.

Note that for all operating modes, if the `DIFS` bit is set, all conditions are overridden. The frame sync is driven off-chip and the data output are zero with the `DERRX` bit set.

Internal Memory DMA Transfers

SPORT DMA provides a mechanism for receiving or transmitting an entire block of serial data before the interrupt is generated. When serial port DMA is not enabled, the SPORT generates an interrupt every time it receives or starts to transmit a data word. The processor's on-chip DMA controller handles the DMA transfer, allowing the processor core to continue running until the entire block of data is transmitted or received. Service routines can then operate on the block of data rather than on single words, significantly reducing overhead.

Therefore, set the direction bit, the serial port enable bit, and DMA Enable bits before initiating any operations on the SPORT data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, it can cause unpredictable results.

Each transmitter and receiver has its own DMA registers. 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 are 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 10-5 on page 10-28](#) 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 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 higher priority causes the SPORT DMA transfers to be performed first when multiple DMA requests occur in the same cycle. The serial port DMA channels are numbered and prioritized as shown in [Table 2-28 on page 2-36](#).

Although the DMA transfers are performed with 32-bit words, serial ports can handle word sizes from 3 to 32 bits, with 8 to 32 bits for 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 serial port data packing is enabled (PACK = 1), the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.

External Memory DMA Transfers

In previous SHARC processors, transferring data from a SPORT to external memory required placing that data temporarily in internal memory

and then transferring it to external memory using DMA. The ADSP-214xx processors allow direct DMA transfers between SPORTs and external memory which removes this overhead, freeing up the core and internal memory for other peripherals. The SPORT DMA and chain pointer registers have been expanded to hold the external memory address.

Standard DMA

Each SPORT DMA channel has an enable bit (`SDEN_A` and `SDEN_B`) in its `SPCTLX` register. When DMA is disabled for a particular channel, the SPORT generates an interrupt every time it receives a data word or whenever there is a vacancy in the transmit buffer. For more information, see “[Single Word Transfers](#)” on page [10-43](#).

To set up a serial port DMA channel, write a set of memory buffer parameters to the SPORT DMA parameter registers as shown in [Table 2-14 on page 2-14](#).

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

Once serial port DMA is enabled, the processor’s DMA controller automatically transfers received data words in the receive buffer to the buffer in internal or external memory, depending on the transfer type. Likewise, when the serial port is ready to transmit data, the DMA controller automatically transfers a word from internal or external memory to the transmit buffer. The controller continues these transfers until the entire data buffer is received or transmitted.

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

DMA Chaining

Each channel also has a DMA chaining enable bit (`SCHEN_A` and `SCHEN_B`) in its `SPCTLx` control register.

Each SPORT DMA channel also has a chain pointer register (`CPSPxy`). The `CPSPxy` register functions are used in chained DMA operations.

In chained DMA operations, the processor's DMA controller automatically sets up another DMA transfer when the contents of the current buffer have been transmitted (or received). The chain pointer register (`CPSPxy`) functions as a pointer to the next set of buffer parameters stored in external or internal memory. The DMA controller automatically downloads these buffer parameters to set up the next DMA sequence. For more information on SPORT DMA chaining, see “[DMA Chaining](#)” on [page 2-32](#).

DMA chaining occurs independently for the transmit and receive channels of each serial port. Each SPORT DMA channel has a chaining enable bit (`SCHEN_A` or `SCHEN_B`) that when set (= 1), enables DMA chaining and when cleared (= 0), disables DMA chaining. Writing all zeros to the address field of the chain pointer register (`CPSPxy`) also disables chaining.



The chain pointer register should be cleared first before chaining is enabled.

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



Although the word lengths can be 3 to 32 bits, transmitting or receiving words smaller than 7 bits at the full clock rate of the serial port may cause incorrect operation when DMA chaining is enabled. Chaining locks the processor's internal I/O bus for several cycles while the new transfer control block (TCB) parameters are being loaded. Receive data may be lost (for example, overwritten) during this period.

Data Transfers

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

DMA Chain Insertion Mode

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

When DMA on a channel is disabled and chaining on the channel is enabled, the DMA channel is in chain insertion mode. This mode allows a program to insert a new DMA or DMA chain within the current chain without effecting the current DMA transfer.

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

Chain insertion should not be set up as an initial mode of operation. This mode should only be used to insert one or more TCBs into an active DMA chaining sequence. For more information, see “[Enter DMA Chain Insertion Mode](#)” on page 10-56.

Frame Sync Generation

The frame syncs are generated if the transmit or receive buffers are updated according to the DIFS bit setting (=0). The SPORT DMA controller ensure that the data buffers are updated accordingly.

If both A and B channels are enabled, one of the following can occur.

- In standard mode the `FS_BOTH` bit (in the `SPCTLx` register) defines the conditions of whether both channels are logically ANDed or ORed.
- For all other operating modes, channels A and B are logically ANDed. If both channels are enabled, both buffers need to be updated by the DMA controller to drive data and frame sync off-chip.

Note that for all operating modes, if the `DIFS` bit is set and the DMA transfers have completed, the frame sync continues to drive off-chip and the data output are zero with the `DERRx` bit set.

Interrupts

This section handles the various scenarios in which an interrupt is triggered. Both the core and DMA are able to generate data interrupts for receive or transmit operations. Moreover, the SPORT modules generate error conditions which generate a separate interrupt.

[Table 10-10](#) provides an overview of SPORT interrupts.

Table 10-10. SPORT Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
SPORT (standard, I2S, left justified, packed, multichannel, 16 channels)	<ul style="list-style-type: none"> – DMA RX/TX done – Core RX buffer full – Core TX buffer empty – DMA under/overflow error – Frame sync error 	Internal transfer or access completion	RTI instruction	P3I-P8I, P11I, P16I, SPERRI

Internal Transfer Completion

Each serial port has an interrupt associated with it. For each SPORT, both the A and B channel transmit and receive data buffers share the same interrupt vector. The interrupts can be used to indicate the completion of the transfer of a block of serial data when the serial ports are configured for DMA. They can also be used to perform single word transfers (refer to “[Single Word Transfers](#)” on page 10-43). The priority of the serial port interrupts is shown in [Table 2-28 on page 2-36](#).

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.



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

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

Each DMA channel has a count register (`CSPxA/CSPxB`), which must be initialized with a word count that specifies the number of words to transfer. The count register decrements after each DMA transfer on the channel. When the word count reaches zero, the SPORT generates an interrupt, then automatically stops the DMA channel.

Shared Channels

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 (DXS_A, DXS_B) in SPCTLx registers and for DMA the corresponding status bits in the SPMCTLx registers. However note in most cases if both channels are enabled with the same DMA count, there is no need to check the status since both channel interrupts are close to each other.



Standard DMA does not function properly in I²S/left-justified mode when two channels (A and B) are enabled with different DMA count values. In this case, the interrupt is generated for the least count only. If both the A and B channels of the SPORTs are used in I²S/left-justified mode with DMA enabled, then the DMA count value should be the same for both channels. This does not apply to chained DMA.

Error Detection

Similar to previous SHARC processors, the SPORTs can return the status of data buffer underflow and overflow conditions. Additionally, the SPORTs can also detect frame syncs that are occurring early, even before the last transmit or receive completes. To detect these errors, the processor has an error interrupt (SPERRI vector interrupt) that is shared for all SPORTs together. It is triggered on a data underflow, data overflow, or frame sync error in their respective channels. An interrupt is triggered and programs simply read the SPERRSTAT register which reduces the processor overhead needed to do register polling. If the interrupt enable bit SPERRI is set then the interrupt is raised when the error occurs. Otherwise, the errors are latched and no interrupt is generated.

As shown in [Figure 10-10](#), the frame sync error (which sets the error bit) is triggered when an early frame sync occurs during data transmission or reception or for late frame sync if the period of the frame sync is smaller than the serial word length (SLEN). However, the current transmit/receive operation continues without interruption.

Interrupts

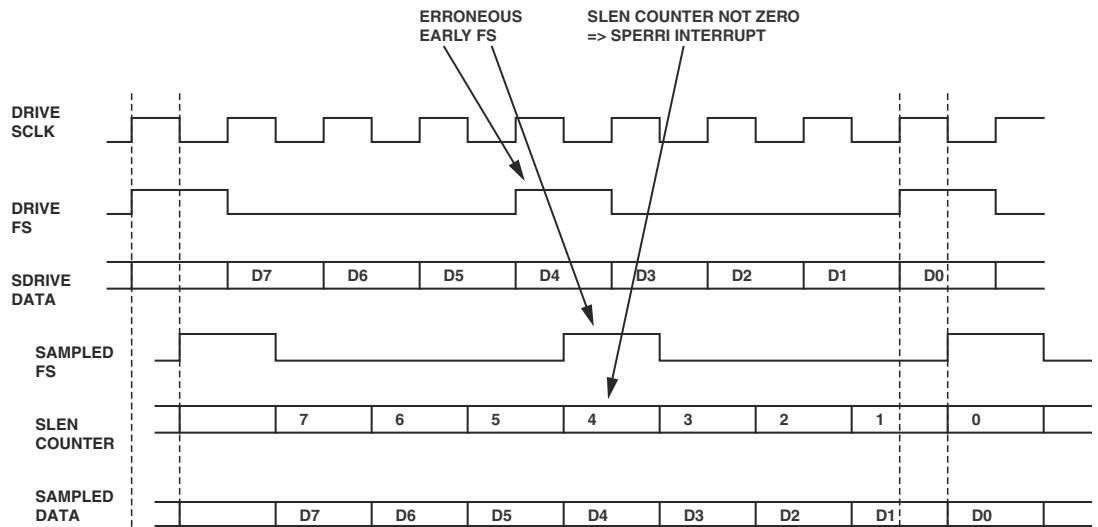


Figure 10-10. Frame Sync Error Detection

When a serial port is receiving or transmitting, its bit count is set to a word length (for example 32 bits). After each clock edge the bit count is decremented. After the word is received/transmitted the bit count reaches zero, and on next frame sync it is set to 32. When active transmission or reception is occurring, the bit count value is non-zero. When a frame sync with a bit count of non-zero is detected, a frame sync error occurs.

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. This is considered a valid frame sync.
- If there is a underflow or overflow error. SPORT error logic does not run (the bit count is not set and decremented) if there is an underflow error. Therefore, frame sync errors can not be detected.

- When the frame sync pulse > SCLK period.
- In late frame sync mode if the frame sync pulse is not active during the whole transmission/reception a frame sync error is generated.

Error Status

Each SPORT can generate an interrupt if a DERR_A, DERR_B, or FSYNC_ERR error occurs. The SPERRCTL_x registers control and report the status of the interrupts generated by each SPORT.

SPORT sticky error bits can be cleared in two ways:

1. By disabling the SPORT (frame sync error) or disabling the corresponding channel by itself (for DERR_A, DERR_B).
2. By writing a 1 to the interrupt status bits in the SPERRCTL_x register. When sticky bits are cleared, interrupts are also cleared.

Only one error interrupt is connected for all serial ports together. So when an error occurs the programs should read the sticky status bits and detect which interrupt caused the error.

An additional register is provided to read all sport interrupt status bits together. The SPERRSTAT register shows the status of all SPORT error interrupts. This register also shows the latched interrupt status, but only when the interrupt is enabled for that error.

Debug Features

The following sections provide information on debugging features available with the serial ports.

Effect Latency

SPORT Loopback

When the SPORT loopback bit, SPL (bit 12), is set in the SPMCTL_x register, the serial port is configured in an internal loopback connection as follows: SPORT0/SPORT1 work as a pair, SPORT2/SPORT3 work as a pair, SPORT4/SPORT5 work as a pair and SPORT6/SPORT7 work as a pair.



The SPL bit applies to all non multichannel modes.

The loopback mode enables programs to test the serial ports internally and to debug applications. In loopback mode, either of the two paired SPORTS can be transmitters or receivers. One SPORT in the loopback pair must be configured as a transmitter; the other must be configured as a receiver. For example, SPORT0 can be a transmitter and SPORT1 can be a receiver for internal loopback. Or, SPORT0 can be a receiver and SPORT1 can be the transmitter when setting up internal loopback.

LoopBack Routing

The SPORTs support an internal loopback mode by using the SRU. [For more information, see “Loop Back Routing” on page 9-40.](#)

Buffer Hang Disable (BHD)

To support debugging buffer transfers, the processors have a buffer hang disable (BHD) bit. When set (= 1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition.

Effect Latency

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

Write Effect Latency

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

SPORT Effect Latency

After a write to a SPORT control register, control and mode bit changes take effect in the second serial clock cycle (SCLK).

The SPORT is ready to start transmitting or receiving three serial clock cycles after they are enabled in the SPCTL_x control register. No serial clocks are lost from this point on. This delay does also apply in slave mode (external clock/frame sync) for synchronization.

Multichannel and packed operation is activated 3 serial clock cycles (SCLK) after the MCEA or MCEB bits are set. Internally-generated frame sync signals activate 4 serial clock cycles after the MCEA or MCEB bits are set.

Programming Model

The section describes some programming procedures that are used to enable and operate the SPORTs.

Setting Up and Starting DMA Master Mode

To set up and initiate a master DMA operation, use the following procedure.

1. Clear the SPORT control register (SPCTL_x).
2. Write to the appropriate DIV_x register, setting the master clock and frame sync ratios.

Programming Model

3. Configure all DMA parameter registers (index, modify and count).
4. Configure the SPORT operation mode and enable DMA operation (SPCTLx).

Setting Up and Starting Chained DMA

To set up and initiate a chain of DMA operations, use the following procedure.

1. Clear the chain pointer register.
2. For internal memory transfers, set up all TCBs in internal memory.
3. Write the address containing the index register value of the first TCB to the chain pointer register, which starts the chain.
4. Write to the SPCTLx register by setting the DMA enable bit to one and the chaining enable bit to one. Setting these bits loads the DMA parameter registers.

Enter DMA Chain Insertion Mode

Chain insertion lets the SPORTs insert a single SPORT DMA operation or another DMA chain within an active SPORT DMA chain.

1. Enter chain insertion mode by setting SDENx = 0 and SCHENx = 1 in the channel's DMA control register. The DMA interrupt indicates when the current DMA sequence is complete.
2. Copy the address currently held in the chain pointer register to the chain pointer position of the last TCB in the chain that is being inserted.

3. Write the start address of the first TCB of the new chain into the chain pointer register.
4. Resume chained DMA mode by setting $\text{SDEN}_x = 1$ and $\text{SCHEN}_x = 1$.

Setting Up and Starting Multichannel Mode

Use the SPCTL_x and channel selection registers (SPMCTL_x) to configure the serial ports to run in multichannel mode as follows. For proper data alignment on sports in multichannel mode, the multichannel enable bit must be set last.

1. Clear all control registers (SPCTL_x/y and SPMTCL_x/y)
2. Configure the channel section registers (SPXCS_x and SPYCS_x).
3. For DMA mode operation, configure the DMA parameter registers (Index, Modify and Count). For DMA chaining, initialize the chain pointer register with the index register for the first chain.
4. Configure the transmitter SPORT_x control register of a SPORT_{xy} pair (SPCTL_x) and enable the DMA/DMA chaining.
5. Configure the receiver SPORT_y control register of pair SPORT_{xy} pair (SPCTL_y) and enable the DMA/DMA chaining.
6. In multichannel and packed I²S modes, the frame sync is independent of data. In multichannel/packed I²S mode, operation starts as soon as the MCE_x bit is enabled.

Due to the priority of other DMA channels, if the DMA controller does not load the transmit buffer with the actual value from memory, then the older value is transmitted out. Therefore, for DMA/DMA chaining mode, wait for the transmit buffer status to

Programming Model

become non-empty by polling the `DXS0_A/B` bits. For core mode operation, initialize the transmit buffer with the first data word to be transmitted.

7. Configure and enable multichannel in the multichannel control registers (`SPMCTLx` and `SPMCTLy`).

Multichannel Mode Backward Compatibility

In previous SHARC models, the serial port pair used the same control register (`SPMCTL01`) to program multichannel mode. In the ADSP-214xx processors, this register is simply renamed to `SPMCTL0` and a new register, `SPMCTL1` has been added. Note that both however are identical. Programs using the older code simply need to change from the `SPMCTL01` register to the `SPMCTL0` register or the `SPMCTL1` register.

The following steps should be taken to port the code to the ADSP-214xx products.

1. Instead of programming `SPMCTLxy` only, program both `SPMCTLx` and `SPMCTLy`.
2. In previous processors the data direction bit in the `SPCTL` register was hard coded in multichannel mode (where the even port is always the transmitter and the odd port is always the receiver). But in the ADSP-214xx processors, the direction (`SPTRAN` bit) is honored and therefore should be set as required.
3. Routing models for hard coded multichannel pairs used the even SPORT for the clock and the odd SPORT for the frame sync. The `TDV` signal was derived from the even frame sync. In the ADSP-214xx processors, these limitations no longer apply. All SPORTs operate completely independently. Therefore every SPORT requires the clock and frame sync to be routed. The `TDV` signal is separate and is fed into the SRU unit.

Programming Packed Mode

Since packed mode is implemented on top of multichannel mode, programming this mode is the same as programming multichannel mode. Use the serial port control (SPCTL_x) and channel selection registers (SPMCTL_x) to configure the serial ports to run in packed mode as follows.

1. Configure the multichannel channel select registers.
2. Set the OPMODE, ICLK, IFS, CKRE bits in the SPCTL_x register to run in packed master mode.
3. Clear the LSBF bit to run in packed mode.
4. To emulate I²S in packed mode, set the MFD bit field to one and the NCH bit field according to the channels in the SPMCTL_x register.

The MFD bit field and the L_FIRST bit allow programs to manipulate the timing as follows.

1. The MFD bit field selects the data delay in SCLK cycles after the frame sync occurred.
2. The L_FIRST bit allows to swap the left and right channels.

Additional Information for External Frame Sync Operation

There are two procedures which allow programs to save SPORT initialization during an inactive frame sync:

- Read the DAI_PIN_STAT register of the frame sync to get the level prior to starting SPORT configuration.
- Route a MISCA register input to the external frame signal (rising or falling edge) as an interrupt trigger to generate an interrupt to start SPORT configuration.



In the ADSP-214xx processors the `FSED` bit in the `SPCTLN` register allows SPORT initialization regardless of the state of the external frame sync. The SPORT starts the transfer on the next valid edge.

Companding As a Function

Since the values in the transmit and receive buffers are actually companded in place, the companding hardware can be used without transmitting (or receiving) any data, for example during testing or debugging. This operation requires one peripheral clock cycle of overhead, as described below. For companding to execute properly, program the SPORT registers prior to loading data values into the SPORT buffers.

To compress data in place without transmitting use the following procedure.

1. Set the `SPTRAN` bit to 1 in the `SPCTLx` register. The `SPEN_A` and `SPEN_B` bits should be = 0.
2. Enable companding in the `DTYPE` field of the `SPCTLx` transmit control register.
3. Write a 32-bit data word to the transmit buffer. Companding is calculated in this cycle.
4. Wait two cycles. Any instruction not accessing the transmit buffer can be used to cause this delay. This allows the serial port companding hardware to reload the transmit buffer with the compounded value.
5. Read the 8-bit compressed value from the transmit buffer.

To expand data in place, use the same sequence of operations with the receive buffer instead of the transmit buffer. When expanding data in this way, set the appropriate serial word length (`SLEN`) in the `SPCTLx` register.

With companding enabled, interfacing the serial port to a codec requires little additional programming effort. If companding is not selected, two formats are available for received data words of fewer than 32 bits—one that fills unused MSBs with zeros, and another that sign-extends the MSB into the unused bits.

Programming Model

11 INPUT DATA PORT

The Input Data Port (IDP) comprises two units: the serial input port (SIP) and the parallel data acquisition port (PDAP). Located inside the DAI of the SHARC processor it provides an efficient way of transferring data from DAI pin buffers, the external port, the asynchronous sample rate converters (ASRC) and the S/PDIF transceiver to the internal memory of SHARC. The IDP specifications are shown in [Table 11-1](#).

Table 11-1. IDP Port Specifications

Feature	SIP	PDAP
Connectivity		
Multiplexed Pinout	No	Yes (External Port)
SRU DAI Required	Yes	Yes
SRU DAI Default Routing	No	No
SRU2 DPI Required	No	No
SRU2 DPI Default Routing	N/A	N/A
Interrupt Control	Yes	Yes
Protocol		
Master Capable	No	No
Slave Capable	Yes	Yes
Transmission Simplex	Yes	Yes
Transmission Half Duplex	No	No
Transmission Full Duplex	No	No

Features

Table 11-1. IDP Port Specifications (Cont'd)

Feature	SIP	PDAP
Access Type		
Data Buffer	Yes	Yes
Core Data Access	Yes	Yes
DMA Data Access	Yes	Yes
DMA Channels	8	1
DMA Chaining	No	No
Boot Capable	No	No
Local Memory	No	No
Clock Operation	$f_{PCLK}/4$	$f_{PCLK}/4$

Features

The following list describes the IDP features.

- The IDP provides a mechanism for a large number of asynchronous channels (up to eight).
- The IDP supports industry standard data formats, I²S, Left-justified and Right-justified for serial input ports.
- The PDAP supports four data packing modes for parallel data.
- The PDAP supports a maximum of 20-bits.
- Provides two data transfer types, through DMA or interrupt driven transfer by core.

Pin Descriptions

[Table 11-2](#) provides descriptions of the IDP pins used for the serial interface port.

Table 11-2. SIP Pin Descriptions

Internal Node	I/O	Description
IDP7_0_CLK_I	I	Serial Input Port Receive Clock Input. This signal must be generated externally and comply to the supported input formats.
IDP7_0_FS_I	I	Serial Input Port Frame Sync Input. The frame sync pulse initiates shifting of serial data. This signal must be generated externally and comply to the supported input formats.
IDP7_0_DAT_I	I	Serial Input Port Data Input. Unidirectional data pin. Data signal must comply to the supported data formats.

[Table 11-3](#) provides descriptions of the IDP pins used for the parallel interface port.

Table 11-3. PDAP Pin Descriptions

Internal Nodes	Type	Description
PDAP_CLK_I	I	Parallel Data Acquisition Port Clock Input. Positive or negative edge of the PDAP clock input is used for data latching depending on the IDP_PDAP_CLKEDGE bit (29) of the IDP_PP_CTL register. Note that input has multiplexed.
PDAP_HOLD_I	I	Parallel Data Acquisition Port Frame Sync Input. The PDAP hold signal determines whether the data is to be latched at an active clock edge or not. When the PDAP hold signal is HIGH, all latching clock edges are ignored and no new data is read from the input pins. The packing unit operates as normal, but it pauses and waits for the PDAP hold signal to be deasserted and waits for the correct number of distinct input samples before passing the packed data to the IDP FIFO. Note that the input has multiplexed control.
PDAP_DATA	I	Parallel Data Acquisition Port Data Input. The PDAP latches 20-bit parallel data which where packed into 32-bits by using different packing. Note that input has multiplexed control.

Pin Descriptions

Table 11-3. PDAP Pin Descriptions (Cont'd)

Internal Nodes	Type	Description
PDAP_STRB_O	O	Parallel Data Acquisition Port Clock input. The PDAP packing unit asserts the output strobe whenever there is 32-bit data available for transfer to the IDP FIFO. The width of this pulse is equal to 2 x PCLK cycles. This signal can be used to synchronize external requests for new PDAP data. Note that input has multiplexed control.

Table 11-4 provides descriptions of the pin multiplexing between DAI and external port. [For more information, see “Pin Multiplexing” on page 23-28.](#)

Table 11-4. Pin Multiplexing between DAI and External Port

Signal	DAI	External Port
Serial Clock	IDP0_CLK_I	DATA[10]
Frame Sync	IDP0_FS_I	DATA[11]
Data	DAI_PB20–1	DATA[31–12]
Strobe Out	PDAP_STRB_O	DATA[8]

SRU Programming

The SRU (signal routing unit) needs to be programmed in order to connect the IDP to the output pins as shown in [Table 11-5](#).

Table 11-5. IDP DAI/SRU Signal Connections

Internal Node	DAI Group	SRU Register
Inputs		
IDP7_0_CLK_I	Group A	SRU_CLK3–2
IDP7_0_FS_I	Group C	SRU_FS3–2
IDP7_0_DAT_I	Group B	SRU_DAT5–4

[Table 11-6](#) shows the signal connections when using the PDAP on the DAI pins.

Table 11-6. PDAP DAI/SRU Signal Connections

Internal Node	DAI Connection	SRU Register
Inputs		
IDP0_CLK_I	Group A	SRU_CLK2
PDAP_HOLD_I	Group C	SRU_FS2
DAI_PB20_1_I	Group D	SRU_PIN4–0
Outputs		
PDAP_STRB_O	Group D	

Register Overview

This section provides brief descriptions of the major registers. For complete information see “[Input Data Port Registers](#)” on page [A-174](#).

Clocking

IDP Control Registers (IDP_CTLx). The ADSP-2136x and ADSP-2137x SHARC processors have two IDP control registers. The IDP_CTL1-0 registers are used to control the SIP operations.

PDAP Control Register (IDP_PP_CTL). The register (shown in Figure 11-1) is used to control all PDAP operations.

IDP Status Register (IDP_STAT). The register returns different types of status for SIP/PDAP core and DMA operations.

Clocking

The fundamental timing clock of the IDP module is peripheral clock/4 ($\text{PCLK}/4$). The IDP SIP/PDAP operates in slave mode only.

Functional Description

The IDP provides up to eight serial input channels—each with its own clock, frame sync, and data inputs. The eight channels are automatically multiplexed into a single 32-bit by eight-deep FIFO. Data is always formatted as a 64-bit frame and divided into two 32-bit words. The serial protocol is designed to receive audio channels in I²S, left-justified, or right-justified mode. One frame sync cycle indicates one 64-bit left-right pair, but data is sent to the FIFO as 32-bit words (that is, one-half of a frame at a time). Transfers from this FIFO to internal memory can be performed either via DMA or by interrupts driven by the core.



IDP Channel 0 is shared by SIP0 and PDAP. All other 7 SIPs are connected to corresponding IDP channel of FIFO.

The DMA engine of the IDP implements DMA for all the 8 channels. It has eight sets of DMA parameter registers for 8 channels. Data from channel 0 is directed to internal memory location controlled by set of registers for channel 0 and so on.

The parallel data is acquired through the parallel data acquisition port (PDAP) which provides a means of moving high bandwidth data to the core's memory space. The data may be sent to memory as one 32-bit word per input clock cycle or packed together (for up to four clock cycles worth of data).

[Figure 11-1](#) provides a graphical overview of the input data port architecture. Notice that each channel is independent and contains a separate clock and frame sync input.



The IDP provides an easy way to pump serial data into on-chip memory since it is less complex than the traditional SPORT module, limited to unidirectional slave transfers only.

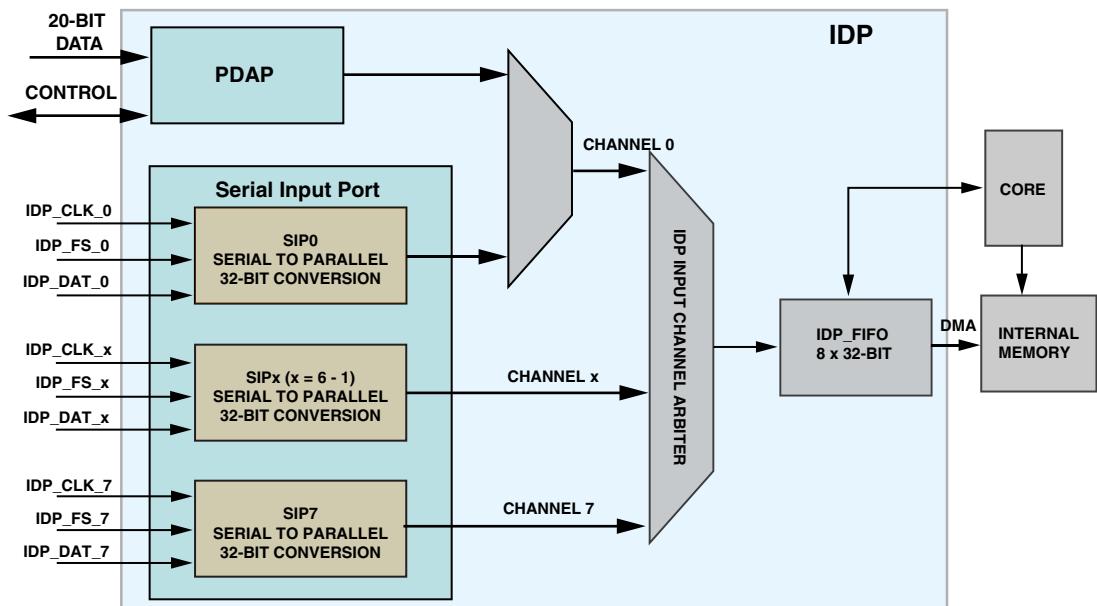


Figure 11-1. Input Data Port

Operating Modes

The following sections provide information on the various operation modes used by the PDAP module. The IDP has access to the IDP FIFO in the three modes listed below. The bit settings that configure these modes are shown in [Table 11-7](#).

- Core mode (SIP/PDAP)
- DMA mode (SIP/PDAP)
- DMA ping pong mode (SIP/PDAP)

Table 11-7. IDP Operation Modes

IDP Operation Modes	IDP_CTL0 Global Control		IDP_CTL1 Channel Control			IDP_PP_CTL	
	IDP_EN	IDP_DMA_EN	IDP_ENx	IDP_DMA_ENx	IDP_PINGx	IDP_PDAP_EN	PDAP_PP_SELECT
Core SIP7-0	1	0	1	0	0	0	0
Core PDAP DAI	1	0	1	0	0	1	0
Core PDAP EP	1	0	1	0	0	1	1
DMA SIP7-0	1	1	1	1	0	0	0
DMA PDAP DAI	1	1	1	1	0	1	0
DMA PDAP EP	1	1	1	1	0	1	1
DMA Ping Pong SIP7-0	1	1	1	1	1	0	0
DMA Ping Pong PDAP DAI	1	1	1	1	1	1	0
DMA Ping Pong PDAP EP	1	1	1	1	1	1	1

PDAP Port Selection

The input to channel 0 of the IDP is multiplexed, and may be used either in the serial mode or in a direct parallel input mode. Setting the `PDAP_EN` bit high disables the connection of SIP0 to channel 0 of the FIFO. The data inputs can come either from the DAI pins or the external port ADDR pins. This is selected by the `PDAP_PP_SELECT` bit in the `PDAP_CTL` register.

[Figure 11-2](#) illustrates the data flow for IDP channel 0, where either the PDAP or serial input can be selected.

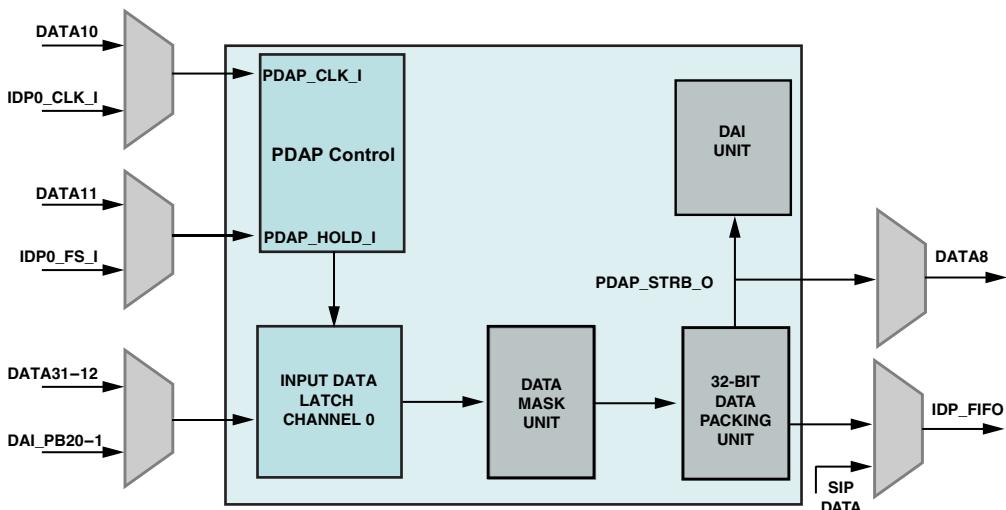


Figure 11-2. PDAP Port (Detail of IDP Channel 0)

Data Hold

When the `PDAP_HOLD` signal is high, all latching clock edges are ignored and no new data is read from the input pins. The packing unit operates as normal, but it pauses and waits for the `PDAP_HOLD` signal to be deasserted and waits for the correct number of distinct input samples before passing the packed data to the FIFO.

Operating Modes

[Figure 11-3 on page 11-11](#) through [Figure 11-5 on page 11-13](#) show different packing modes including valid data hold inputs.

As shown in the figures, PDAP_DATA and PDAP_HOLD are driven by the inactive edges of the clock (falling edge in the above figures) and these signals are sampled by the active edge of the clock (rising edge in the figures).

PDAP Data Masking

For input data widths less than 20, inputs are aligned to the MSB pins. Additionally all PDAP inputs can be masked (`IDP_PDAP_CTL` register) to form user specific data streams from any input pins. Clearing the MASK bits (=0) disables data from the corresponding DAI or external port pin.

PDAP Data Packing

Multiple latched parallel sub word samples may be packed into 32-bit words for efficiency. The frame sync input is used to hold off latching of the next sample (that is, ignore the clock edges). The data then flows through the FIFO and is transferred by a dedicated DMA channel into the core's memory as with any IDP channel. As shown in [Figure 11-2](#), the PDAP can accept input words up to 20 bits wide, or can accept input words that are packed as densely as four input words up to eight bits wide.

The `IDP_PDAP_PACKING` bits define the packing format. Based on the PDAP packing the data buffer format changes as shown in [Figure 11-9](#).

No Packing

No packing provides for 20 bits coming into the packing unit and 32 bits going out to the FIFO in a single cycle. On every clock edge, 20 bits of data are moved and placed in a 32-bit register, left-aligned. That is, bit 19 maps to bit 31. The lower bits, 11–0, are always set to zero.

This mode sends one 32-bit word to FIFO for each input clock cycle—the DMA transfer rate matches the PDAP input clock rate.

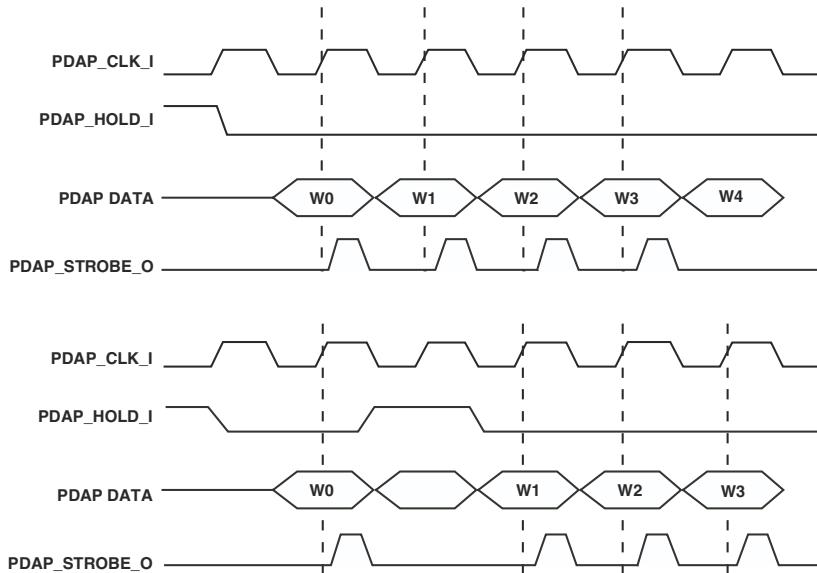


Figure 11-3. PDAP Hold Input (No Packing)

Packing by 2

Packing by 2 moves data in two cycles. Each input word can be up to 16 bits wide.

- On clock edge 1, bits 19–4 are moved to bits 15–0 (16 bits)
- On clock edge 2, bits 19–4 are moved to bits 31–16 (16 bits)

This mode sends one packed 32-bit word to FIFO for every two input clock cycles—the DMA transfer rate is one-half the PDAP input clock rate.

Operating Modes

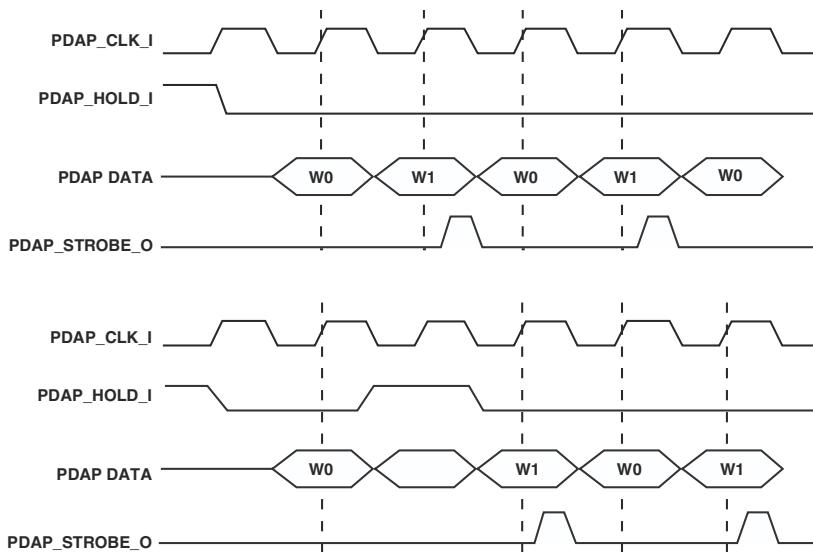


Figure 11-4. PDAP Hold Input (Packing by 2)

Packing by 3

Packing by 3 packs three acquired samples together. Since the resulting 32-bit word is not divisible by three, up to ten bits are acquired on the first clock edge and up to eleven bits are acquired on each of the second and third clock edges:

- On clock edge 1, bits 19–10 are moved to bits 9–0 (10 bits)
- On clock edge 2, bits 19–9 are moved to bits 20–10 (11 bits)
- On clock edge 3, bits 19–9 are moved to bits 31–21 (11 bits)

This mode sends one packed 32-bit word to FIFO for every three input clock cycles—the DMA transfer rate is one-third the PDAP input clock rate.

Packing by 4

Packing by 4 moves data in four cycles. Each input word can be up to eight bits wide.

- On clock edge 1, bits 19–12 are moved to bits 7–0
- On clock edge 2, bits 19–12 are moved to bits 15–8
- On clock edge 3, bits 19–12 are moved to bits 23–16
- On clock edge 4, bits 19–12 are moved to bits 31–24

This mode sends one packed 32-bit word to FIFO for every four input clock cycles—the DMA transfer rate is one-quarter the PDAP input clock rate.

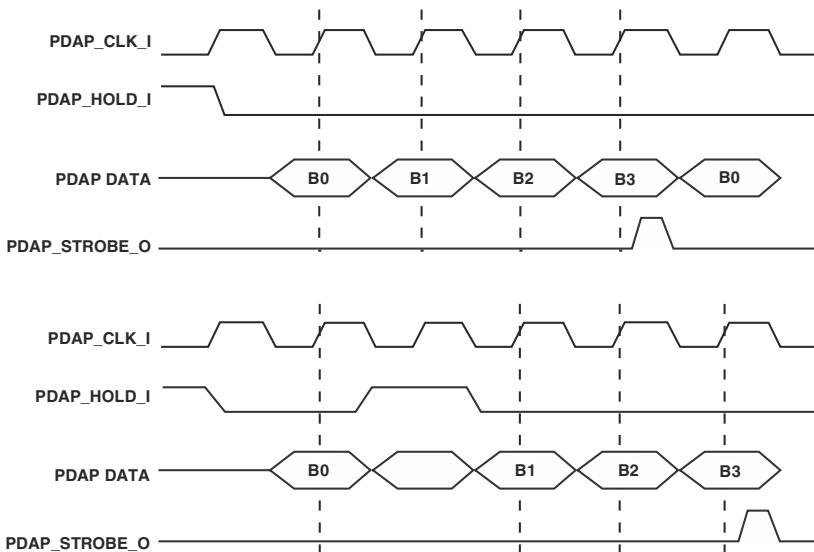


Figure 11-5. PDAP Hold Input (Packing by 4)

Data Transfer

The data from each of the eight IDP channels is inserted into an eight register deep FIFO, which can only be transferred to the core's memory space sequentially. Data is moved into the FIFO as soon as it is fully received. One of two methods can be used to move data from the IDP FIFO to internal memory:

- The core can remove data from the FIFO manually. This method of moving data from the IDP FIFO is described in the next section, “Core Transfers” on page 11-15.
- Eight dedicated DMA channels can sort and transfer data. This method of moving data from the IDP FIFO is described in “DMA Transfers” on page 11-19.

Data Buffer

The `IDP_FIFO` register provides information about the output of the 8-deep IDP FIFO which have been filled by the SIP or the PDAP units. Normally, this register is used only to read and remove the top sample from the FIFO. Channel encoding provides for eight serial input types that correspond to the `IDP_SMODEX` bits in the IDP control registers. When using channels 0–7 in serial mode, this register format applies. When using channel 0 in parallel mode, refer to the description of the packing bits for PDAP mode.



The information in Table 11-8 is not valid when data comes from the PDAP channel.

Table 11-8. IDP_FIFO Register Bit Descriptions

Bit	Name	Description
2–0	CHAN_ENC	IDP Channel Encoding. These bits indicate the serial input port channel number that provided this serial input data. Note: This information is not valid when data comes from the PDAP.
3	LR_STAT	Left/Right Channel Status. Indicates whether the data in bits 31–4 is the left or the right audio channel as dictated by the frame sync signal. The polarity of the encoding depends on the serial mode selected in IDP_SMODE for that channel. See Table A-90 on page A-176 .
31–4	SDATA	Input Data (Serial). Some LSBs can be zero, depending on the mode.

Core Transfers

The core transfers require that the serial peripheral at the SIP writes data to the `IDP_DATAx_I` pin (DATA or DAI pins for PDAP) according to the selected input format used. These data are automatically moved to the `IDP_FIFO` register without DMA intervention.

The output of the FIFO can be directly fetched by reading from the `IDP_FIFO` buffer. The `IDP_FIFO` buffer is used only to read and remove the top sample from the FIFO, which is a maximum of eight locations deep. When this register is read, the corresponding element is removed from the IDP FIFO, and the next element is moved into the `IDP_FIFO` register. A mechanism is provided to generate an interrupt when more than a specified number of words are in the FIFO. This interrupt signals the core to read the `IDP_FIFO` register.

The number of data samples in the FIFO at any time is reflected in the `IDP_FIFOSZ` bit field (bits 31–28 in the `DAI_STAT0` register), which tracks the number of samples in FIFO.

Data Transfer

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.

- i** The maximum data transfer width to internal memory is 32-bits, as in the case of PDAP data or I²S and left-justified modes in single channel mode using 32 bits of data. Therefore, PDAP or I²S and left-justified 32-bit modes cannot be used with other channels in the core/interrupt driven mode since no channel information is available in the data stream.

SIP Data Buffer Format

An audio signal that is normally 24 bits wide is contained within the 32-bit word. Four bits are available for status and formatting data (compliant with the IEC 90958, S/PDIF, and AES3 standards). An additional bit identifies the left-right one-half of the frame. If the data is not in IEC standard format, the serial data can be any data word up to 28 bits wide. Unlike DMA, the core requires a status information about which channel triggered the interrupt. It does this by reading the data buffer. The remaining three bits are used to encode one of the eight channels being passed through the FIFO to the core. The FIFO output may feed eight DMA channels, where the appropriate DMA channel (corresponding to the channel number) is selected automatically.

- i** Regardless of mode, the L/R channel status bit (Bit 3) always specifies whether the data is received in the left channel or the right channel of the corresponding input frame, as shown in [Figure 11-6](#).

AUDIO DATA	INVALID BITS	L/R	IDP CHNL
31	8 7	4 3 2	0

Figure 11-6. Principle Data Format for the SIP

Note that each input channel has its own clock and frame sync input, so unused IDP channels do not produce data and therefore have no impact on FIFO throughput. The clock and frame sync of any unused input should be routed by the SRU to low to avoid unintentional acquisition.

The framing format is selected by using the `IDP_SMODEX` bits (three bits per channel) in the `IDP_CTL0` register. Bits 31–8 of the `IDP_CTL0` register control the input format modes for each of the eight channels. The eight groups of three bits indicate the mode of the serial input for each of the eight IDP channels.

[Figure 11-8](#) and [Figure 11-7](#) shows the IDP data buffer input format for the SIP (depending on `SMODEX` bits) for core access.

RIGHT-JUSTIFIED FORMAT, 24-BIT DATA WIDTH

24 BITS AUDIO DATA	4 BITS, SET TO ZERO	L/R BIT	3 BITS IDP CHANNEL
--------------------	---------------------------	------------	--------------------------

RIGHT-JUSTIFIED FORMAT, 20-BIT DATA WIDTH

20 BITS AUDIO DATA	8 BITS, SET TO ZERO	L/R BIT	3 BITS IDP CHANNEL
--------------------	---------------------------	------------	--------------------------

RIGHT-JUSTIFIED FORMAT, 18-BIT DATA WIDTH

18 BITS AUDIO DATA	10 BITS, SET TO ZERO	L/R BIT	3 BITS IDP CHANNEL
--------------------	----------------------------	------------	--------------------------

RIGHT-JUSTIFIED FORMAT, 16-BIT DATA WIDTH

16 BITS AUDIO DATA	12 BITS, SET TO ZERO	L/R BIT	3 BITS IDP CHANNEL
--------------------	----------------------------	------------	--------------------------

Figure 11-7. IDP Data Buffer Format SIP – Right-Justified

Data Transfer

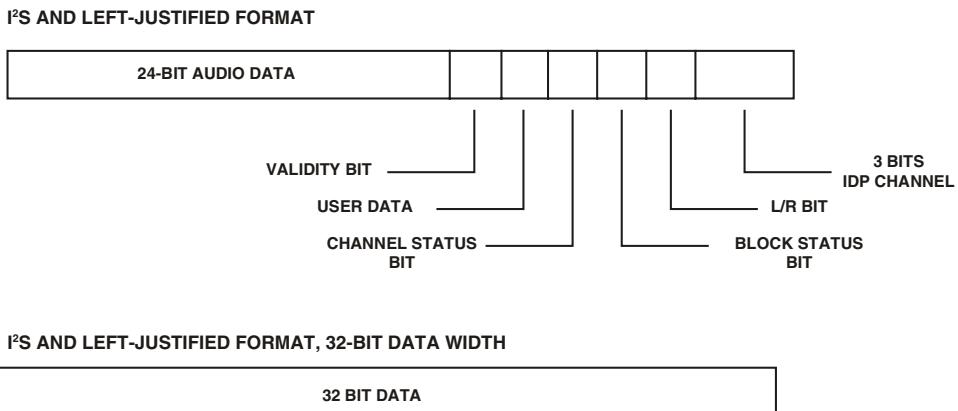


Figure 11-8. IDP Data Buffer Format SIP – I2S/Left-Justified (32 Bits)

The polarity of left-right encoding is independent of the serial mode frame sync polarity selected in `IDP_SMODE` for that channel ([Table 11-3 on page 11-3](#)). Note that I²S mode uses a LOW frame sync (left-right) signal to dictate the first (left) channel, and left-justified mode uses a HIGH frame sync (left-right) signal to dictate the first (left) channel of each frame. In either mode, the left channel has bit 3 set (= 1) and the right channel has bit 3 cleared (= 0).

PDAP Data Buffer Format

If the PDAP module is enabled the IDP data buffer format will change according to the PDAP packing bits (`IDP_PDAP_CTL` register) as shown in [Figure 11-9](#).

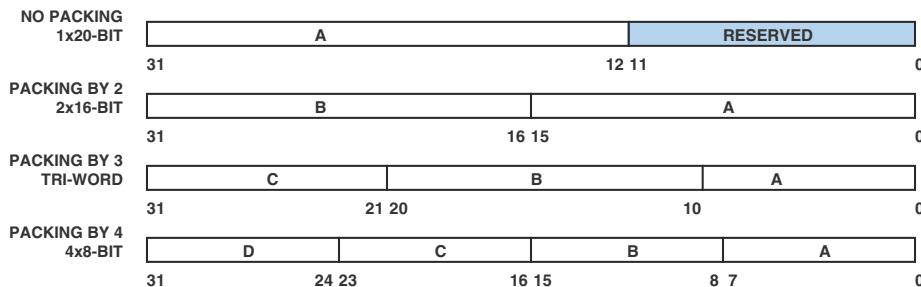


Figure 11-9. IDP Data Buffer Formats for the PDAP

DMA Transfers

The processors support two types of DMA transfers, standard and ping-pong. Eight dedicated DMA channels can sort and transfer the data into one buffer per source channel. When the memory buffer is full, the DMA channel raises an interrupt in the DAI interrupt controller.

Data Buffer Format for DMA

The LSB bits 2–0 of the data format from the serial inputs are channel encoding bits. Since the data is placed into a separate buffer for each DMA channel (defined by parameter index registers), these bits are not required and are cleared (=0) when transferring data to internal memory through the DMA. However, bit 3 still contains the left/right status information. In the case of PDAP data or 32-bit I²S and left-justified modes, these three bits are a part of the 32-bit data.

For serial input channels, data is received in an alternating fashion from left and right channels. Data is not pushed into the FIFO as a full left/right frame. Rather, data is transferred as alternating left/right words as it is received. For the PDAP and 32-bit (non-audio) serial input, data is transferred as packed 32-bit words.

DMA Channel Priority

When more than one channel has data ready, the channels always access the IDP_FIFO register with fixed priority, from low to high channel number (that is, channel 0 is the highest priority and channel 7 is the lowest priority). For the I/O processor, the eight DMA channels are considered as a group and arbitration can rotate across groups for system balance. [For more information, see “Rotating DMA Channel Arbitration” on page 2-44.](#)

Standard DMA

The eight DMA channels each have a set of registers for standard DMA: an I-register, an M-register and a C-register are used.

The IDP DMA parameter registers have these functions:

- **Internal index registers** (`IDP_DMA_Ix`). Index registers provide an internal memory address, acting as a pointer to the next internal memory location where data is to be written.
- **Internal modify registers** (`IDP_DMA_Mx`). Modify registers provide the signed increment by which the DMA controller post-modifies the corresponding internal memory Index register after each DMA write.
- **Count registers** (`IDP_DMA_Cx`). Count registers indicate the number of words remaining to be transferred to internal memory on the corresponding DMA channel.

This DMA access is enabled when the `IDP_EN` bit and `IDP_DMA_EN` bit and the `IDP_DMA_ENx` bits register are set to select a particular channel. The DMA is performed according to the parameters set in the various DMA registers and IDP control registers. An interrupt is generated after end of DMA transfer (when the count = 0).

Ping-Pong DMA

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

The IDP DMA parameter registers have these functions:

- **Internal index registers** (`IDP_DMA_IxA`, `IDP_DMA_IxB`). Index A/B registers provide an internal memory address, acting as a pointer to the next internal memory location where data is to be written.
- **Internal modify registers** (`IDP_DMA_Mx`). Modify registers provide the signed increment by which the DMA controller post-modifies the corresponding internal memory Index register after each DMA write.
- **Ping-Pong Count registers** (`IDP_DMA_Cx`). Count registers indicate the number of words remaining to be transferred to internal memory on the corresponding DMA channel.

This mode is activated when the `IDP_EN` bit, the `IDP_DMA_EN` bit, the `IDP_DMA_ENx` bits, and the `IDP_PINGx` bits are set for a particular channel. An interrupt is generated after every ping and pong DMA transfer (when the count = 0).



Note that ping-pong DMA is repeated until stopped by resetting the `IDP_DMA_ENx` bits (OR global `IDP_DMA_EN` bit).

Multichannel DMA Operation

The SIP/PDAP can run both standard and ping-pong DMAs in different channels. When running standard DMA, initialize the corresponding `IDP_DMA_Ix`, `IDP_DMA_Mx` and `IDP_DMA_Cx` registers. When running

Data Transfer

ping-pong DMA, initialize the corresponding `IDP_DMA_IxA`, `IDP_DMA_IxB`, `IDP_DMA_Mx` and `IDP_DMA_PCx` registers.

DMA transfers for all 8 channels can be interrupted by changing the `IDP_DMA_EN` bit in the `IDP_CTL0` register. None of the other control settings (except for the `IDP_EN` bit) should be changed. Clearing the `IDP_DMA_EN` bit ($= 0$) does not affect the data in the FIFO, it only stops DMA transfers. If the IDP remains enabled, an interrupted DMA can be resumed by setting the `IDP_DMA_EN` bit again. But resetting the `IDP_EN` bit flushes the data in the FIFO. If the bit is set again, the FIFO starts accepting new data.

Programs can drop DMA requests from the FIFO if needed. If one channel has finished its DMA, and the global `IDP_DMA_EN` bit is still set ($=1$), any data corresponding to that channel is ignored by the DMA machine. This feature is provided to avoid stalling the DMA of other channels, which are still in an active DMA state. To avoid data loss in the finished channel, programs can clear ($=0$) `IDP_DMA_EN` bit as discussed in previously.

Multichannel FIFO Status

The state of all eight DMA channels is reflected in the `IDP_DMx_STAT` bits (bits 24–17 of `DAI_STAT` register). These bits are set once the `IDP_DMA_EN` and `IDP_DMA_ENx` bits are set, and remain set until the last data from that channel is transferred. Even if `IDP_DMA_EN` and `IDP_DMA_ENx` bits remain set, the `IDP_DMx_STAT` bits clear once the required number of data transfers takes place.



Note that when a DMA channel is not used (that is, parameter registers are at their default values), the DMA channel's corresponding `IDP_DMx_STAT` bit is cleared ($= 0$).

If the combined data rate from the channels is more than the DMA can service, a FIFO overflow occurs. This condition is reflected for each channel by the individual overflow bits (`SRU_OVFX`) in the `DAI_STAT0` register.

These are sticky bits that must be cleared by writing to the `IDP_CLROVR` bit (bit 6 of the `IDP_CTL0` register). When an overflow occurs, incoming data from IDP channels is not accepted into the FIFO, and data values are lost. New data is only accepted once space is again created in the FIFO.

Interrupts

This section describes the different types of interrupts used by the interface. [Table 11-9](#) provides an overview of IDP interrupts.

Table 11-9. IDP Interrupt Overview

Interrupt Sources	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
DAI IDP (I2S, left/right justified, TDM, 8 channels)	<ul style="list-style-type: none"> – DMA RX done – Core RX buffer size exceeded – RX buffer overflow error 	Internal transfer completion	Read-to-clear <code>DAI_IMASK_x</code> + RTI instruction	P0I, P12I

Interrupt Acknowledge

The correct handling of the IDP interrupt requires that the ISR must read the `DAI_IMASK_x` register to clear the interrupt latch appropriately. Note that many interrupts are combined in the DAI interrupt. Refer to “[Interrupts](#)” on page [9-32](#).

Threshold Interrupts

When using the interrupt scheme, the `IDP_NSET` bits (bits 3-0 of the `IDP_CTL0` register) can be set to N, so N + 1 data can be read from the FIFO in the interrupt service routine (ISR). The `IDP_FIFO_GTN_INT` bit in `DAI_IMASK_X` register allows to fire flexible interrupts in order to respond with the core under different system conditions.

Interrupts

DMA Interrupts

Using DMA transfer overrides the mechanism used for interrupt-driven core reads from the FIFO. When the `IDP_DMA_EN` bit and at least one `IDP_DMA_ENx` of the `IDP_CTL1` register are set, the eighth interrupt (`IDP_FIFO_GTN_INT`) in the `DAI_IMASK_x` registers is NOT generated.

At the end of the DMA transfer for individual channels, interrupts are generated. These interrupts are generated after the last DMA data from a particular channel has been transferred to memory. These interrupts (`IDP_DMAX_INT`) are mapped from the bits 17–10 in the `DAI_IMASK_x` registers and generate interrupts when they are set (= 1). These bits are ORed and reflected in high level interrupts that are sent to the DAI interrupt.

An interrupt is generated at the end of a DMA, which is cleared by reading the `DAI_IMASK_x` registers.

FIFO Overflow Interrupts

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, an interrupt is generated if the `IDP_FIFO_OVR_INT` bit in the `DAI_IMASK_x` register is set (sticky bits in `DAI_STAT0` register are also set). Data is accepted again when space has been created in the FIFO.

Note that the total FIFO depth per channel is 9 locations: 1 location for SIP to parallel data conversion + 8 locations for the `IDP_FIFO`.

Debug Features

The following sections describe the features available for debugging the IDP.

Status register Debug

The core may also write to the FIFO. When it does, the audio data word is pushed into the input side of the FIFO (as if it had come from the SRU on the channel encoded in the three LSBs). This can be useful for verifying the operation of the FIFO, the DMA channels, and the status portions of the IDP. The `IDP_STAT1` register returns the current state of the read/write index pointers from FIFO.

Buffer Hang Disable

The `IDP_BHD` bit in `IDP_CTL0` is used for buffer hang disable control. When there is no data in the FIFO, reading the `IDP_FIFO` register causes the core to hang. This condition continues until the FIFO contains valid data. Setting the `IDP_BHD` bit (= 1) prevents the core from hanging on reads from an empty `IDP_FIFO` register. Clearing this bit (= 0) causes the core to hang under the conditions described previously.

If the `IDP_BHD` bit (bit 4 in the `IDP_CTL0` register) is not set, attempts to read more data than is available in the FIFO results in a core hang.

Shadow Registers

The DAI interrupt controller contains shadow registers to simplify debug techniques since these register are not updated. A read of the `DAI_IMASK_x_SH` register provides the same data as a read of the `DAI_IMASK_x` register. Reading these DAI shadow registers (`DAI_IMASK_x_SH`) does not destroy the contents of the `DAI_IMASK_x` registers.

Effect Latency

Core FIFO Write

The core may also write to the FIFO. When it does, the audio data word is pushed into the input side of the FIFO (as if it had come from the SRU on the channel encoded in the three LSBs). This can be useful for verifying the operation of the FIFO, the DMA channels, and the status portions of the IDP. The `IDP_STAT1` register returns the current state of the read/write index pointers from FIFO.

Effect Latency

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

Write Effect Latency

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

IDP Effect Latency

The IDP is ready to start receiving data one serial clock cycle (`SCLK`) after it is enabled by setting `IDP_EN` bit. No `LRCLK` edges are lost from this point on.

Disabling IDP DMA by resetting the `IDP_DMA_EN` bit requires 1 `PCLK` cycle. Disabling an individual DMA channel by resetting the `IDP_DMA_ENx` bit requires 2 `PCLK` cycles.

Programming Model

The following sections provide procedures that are helpful when programming the input data port.

Setting Miscellaneous Bits

This sequence is used in most following programming models as intermediate step.

Set the required values for:

- IDP_SMODE_x bits in the IDP_CTL_x register to specify the frame sync format for the serial inputs (left-justified I²S, or right-justified mode).
- IDP_Pxx_PDAPMASK bits in the IDP_PP_CTL register to specify the input mask, if the PDAP is used.
- IDP_PP_SELECT bits in the IDP_PP_CTL register to specify input from the DAI pins or the DATA pins, if the PDAP is used.
- IDP_PDAP_CLKEDGE bit (bit 29) in the IDP_PP_CTL register to specify if data is latched on the rising or falling clock edge, if the PDAP is used.

Starting Core Interrupt-Driven Transfer

To start a core interrupt-driven data transfer:

1. Clear the FIFO by setting (= 1) IDP_FFCLR bit (bit 31 in the IDP_CTL1 register).
2. Keep the SCLK and frame sync inputs of the SIP and PDAP connected to low, by setting the proper values in the SRU registers.
3. Refer to “[Setting Miscellaneous Bits](#)” above.
4. Program the SRU registers to establish the proper connection to the SIP and/or PDAP being used. Keep the unused clock and frame sync signals connected to low.

Programming Model

5. Set the desired values for the N_SET variable using the `IDP_NSET` bits in the `IDP_CTL0` register.
6. Set the `IDP_FIFO_GTN_INT` bit (bit 8 of the `DAI_IMASK_RE` register) to HIGH and set the corresponding bit in the `DAI_IMASK_FE` register to LOW to unmask the interrupt. Set bit 8 of the `DAI_IMASK_PRI` register (`IDP_FIFO_GTN_INT`) as needed to generate a high priority or low priority core interrupt when the number of words in the FIFO is greater than the value of N set.
7. Enable the PDAP by setting `IDP_PDAP_EN` (bit 31 in the `IDP_PP_CTL` register), if required.
8. Enable the IDP by setting the `IDP_EN` bit (bit 7 in the `IDP_CTL0` register) and the `IDP_ENx` bits in the `IDP_CTL1` register.



In older SHARC processors, the IDP starts shifting data before the IDP is enabled. However, the shifted data is latched at the next frame sync edge only if the IDP is enabled. Therefore, whether the first channel received by the IDP is left/right depends on the instant when the IDP is enabled—which may lead to channel swapping.

Additional Notes

When IDPs are used to receive data from external devices, there is a sequence to be followed to enable the IDP ports when configured to receive data in I²S mode. Failing to follow this sequence can give rise to channel shift or swap.

1. Connect the frame sync internally using the SRU (Signal Routing Unit) to the DAI interrupt.
2. Configure the DAI interrupt (MISCA) for the inactive edge of the frame sync.

3. Wait for the DAI interrupt, and enable the IDP port inside the DAI interrupt service routine.
4. Clear the DAI interrupt by reading the DAI interrupt latch register. This procedure ensures that the IDP ports are enabled at the correct time, avoiding issues like channel shift or swap in the received data.

Starting A Standard DMA Transfer

To start a DMA transfer from the FIFO to memory:

1. Clear the FIFO by setting (= 1) the `IDP_FFCLR` bit (bit 31 in the `IDP_CTL1` register).
2. While the global `IDP_DMA_EN` and the `IDP_EN` bits are cleared (= 0), set the values for the DMA parameter registers that correspond to channels 7–0.
3. Keep the clock and the frame sync input of the serial inputs and/or the PDAP connected to low, by setting proper values in the SRU registers.
4. Refer to “[Setting Miscellaneous Bits](#)” above.
5. Route all of the required inputs to the IDP by writing to the SRU registers
6. Enable the channel’s `IDP_ENx` and `IDP_DMA_ENx` bit settings.
7. Start the DMA by setting
 - The `IDP_PDAP_EN` bit (bit 31 in `IDP_PP_CTL` register if the PDAP is required).
 - The global `IDP_DMA_EN` bit of the `IDP_CTL0` register to enable standard DMA on the selected channel.

Programming Model

- The global `IDP_EN` bit (bit 7 in the `IDP_CTL0` register).

Starting a Ping-Pong DMA Transfer

To start a ping-pong DMA transfer from the FIFO to memory:

1. Clear the FIFO by setting (= 1) the `IDP_FFCLR` bit (bit 31 in the `IDP_CTL1` register).
2. While the global `IDP_DMA_EN` and `IDP_EN` bits are cleared (=0), set the values for the following DMA parameter registers that correspond to channels 7–0.
3. Keep the clock and the frame sync input of the serial inputs and/or the PDAP connected to LOW, by setting proper values in the SRU registers.
4. Refer to “[Setting Miscellaneous Bits](#)” above.
5. Connect all of the required inputs to the IDP by writing to the SRU registers.
6. Enable the channel’s `IDP_ENx`, `IDP_DMA_ENx` and `IDP_PINGx` bit settings.
7. Start DMA by setting:
 - The `IDP_PDAP_EN` bit (bit 31 in `IDP_PP_CTL` register if the PDAP is required).
 - The global `IDP_DMA_EN` bit of the `IDP_CTL0` register to enable the standard DMA of the selected channel.
 - The global `IDP_EN` bit (bit 7 in the `IDP_CTL0` register).

Servicing Interrupts for DMA

The following steps describe how to handle an IDP ISR for DMA.

1. An interrupt is generated and program control jumps to the ISR when the DMA for a channel completes.
2. The program clears the `IDP_DMA_EN` bit in the `IDP_CTL` register.
 - a. To ensure that the DMA of a particular IDP channel is complete, (all data is transferred into internal memory) wait until the `IDP_DMAX_STAT` bit of that channel becomes zero in the `DAI_STAT` register. This is required if a high priority DMA (for example a SPORT DMA) is occurring at the same time as the IDP DMA.
 - b. As each DMA channel completes, a corresponding bit in either the `DAI_IMASK_L` or `DAI_IMASK_H` register for each DMA channel is set (`IDP_DMAX_INT`).
3. The program clears (= 0) the channel's `IDP_DMA_ENx` bit in the `IDP_CTL1` register which has finished.
4. Reprogram the DMA registers for finished DMA channels.

More than one DMA channel may have completed during this time period. For each, a bit is latched in the `DAI_IMASK_L` or `DAI_IMASK_H` registers. Ensure that the DMA registers are reprogrammed. If any of the channels are not used, then its clock and frame sync should be held LOW.

5. Read the `DAI_IMASK_L` or `DAI_IMASK_H` registers to see if more interrupts have been generated.
 - If the value(s) are not zero, repeat step 4.
 - If the value(s) are zero, continue to step 6.

6. Re-enable the IDP_DMA_EN bit in the IDP_CTL register (set to 1).
7. Exit the ISR.

If a zero is read in step 5 (no more interrupts are latched), then all of the interrupts needed for that ISR have been serviced. If another DMA completes after step 5 (that is, during steps 6 or 7), as soon as the ISR completes, the ISR is called again because the OR of the latched bits will not be nonzero again. DMAs in process run to completion.



If step 5 is not performed, and a DMA channel expires during step 4, then, when IDP DMA is re-enabled, (step 6) the completed DMA is *not* reprogrammed and its buffer overruns.

This unit is multiplexed with SIP0. The PDAP provides one clock input, one clock hold input and a maximum of 20 parallel data input pins. The positive or negative edge of the clock input is used for data latching. The clock hold input (PDAP_HLD_I) validates a clock edge—if this input is high then clock edge is masked for data latching. It supports four types of data packing mode selected by MODE bits in the PDAP_CTL register.

12 ASYNCHRONOUS SAMPLE RATE CONVERTER

The asynchronous sample rate converter (SRC) block is used to perform synchronous or asynchronous sample rate conversion across independent stereo channels, without using any internal processor resources. Furthermore, the SRC is used to clean up audio data from jittery clock sources such as the S/PDIF receiver. The SRC specifications are listed in [Table 12-1](#).

Table 12-1. SRC Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	Yes
SRU DAI Default Routing	No
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	No
Slave Capable	Yes
Transmission Simplex	Yes
Transmission Half Duplex	No
Transmission Full Duplex	No

Features

Table 12-1. SRC Specifications (Cont'd)

Feature	Availability
Access Type	
Data Buffer	No
Core Data Access	N/A
DMA Data Access	N/A
DMA Channels	N/A
DMA Chaining	N/A
Boot Capable	N/A
Local Memory	Yes (RAM, ROM)
Clock Operation	$f_{PCLK}/4$

Features

The SRC for the SHARC processors has the features shown in the list below.

- 4 sample rate converters
- Automatically senses sample frequencies
- Simple programming required
- Attenuates sample clock jitter
- Supports left-justified, I²S, right-justified (16-,18-, 20-, 24-bits), and TDM serial port (daisy chain) modes
- Accepts 16-/18-/20-/24-bit data
- Up to 192 kHz sample rate input/output sample ratios from 7.75:1 to 1:8

- Linear phase FIR filter
- Controllable soft mute

Pin Descriptions

The SRC has two interfaces: an input port and an output port. [Table 12-2](#) describes the six inputs and two outputs for the IP (input port) and OP (output port).

Table 12-2. SRC Pin Descriptions

ADSP-214xx Internal Node	I/O	Description
SRC3_0_CLK_IP_I	Input	SRC input port clock input
SRC3_0_FS_IP_I	Input	SRC input port frame sync input
SRC3_0_DAT_IP_I	Input	SRC input port data input
SRC3_0_CLK_OP_I	Input	SRC output port clock input
SRC3_0_FS_OP_I	Input	SRC output port frame sync input
SRC3_0_TDM_OP_I	Input	SRC output port TDM daisy chain data input
SRC3_0_DAT_OP_O	Output	SRC output port data output
SRC3_0_TDM_IP_O	Output	SRC output port TDM daisy chain data output

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 12-3](#).

Register Overview

Table 12-3. SRC DAI/SRU Signal Routing

ADSP-214xx Internal Node	DAI Connection	SRU Register
Inputs		
SRC3-0_CLK_IP_I SRC3-0_CLK_OP_I	Group A	SRU_CLK2-1
SRC3-0_FS_IP_I SRC3-0_FS_OP_I	Group C	SRU_FS2-1
SRC3-0_DAT_IP_I SRC3-0_TDM_OP_I	Group B	SRU_DAT3-2
Outputs		
SRC3-0_DAT_OP_O	Group B, D	
SRC3-0_TDM_IP_O	Group B	

For information on using the SRU, see “[Rules for SRU Connections](#)” on [page 9-20](#).

Register Overview

The SRC uses five registers to configure and operate the SRC module. For complete register and bit descriptions, see “[Sample Rate Converter Registers](#)” on [page A-184](#).

Control Registers (SRCCTLx). The SRCCTLx registers enable or disable the sample rate converters. They also specify the input and output data format.

Mute Register (SRCMUTE). The SRCMUTE register controls the connection of the mute in and mute out signal.

Ratio Registers (SRCRATx). The SRCRATx registers return the sample ratio between the input and out data stream and mute information (mute out).

Clocking

The fundamental timing clock of the ASRC module is peripheral clock/4 ($\text{PCLK}/4$) and is operating in slave mode only.

Functional Description

[Figure 12-1](#) shows a top level block diagram of the SRC module and [Figure 12-2](#) 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 `SRCx_FS_IP` 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 `SRCx_FS_IP` and `SRCx_FS_OP` sample rates and provides the RAM and ROM start addresses for the start of the FIR filter convolution.



Unlike other peripherals, the sample rate converters own local memories (RAM and ROM) which are dedicated for the purpose of sample rate conversion only.

Functional Description

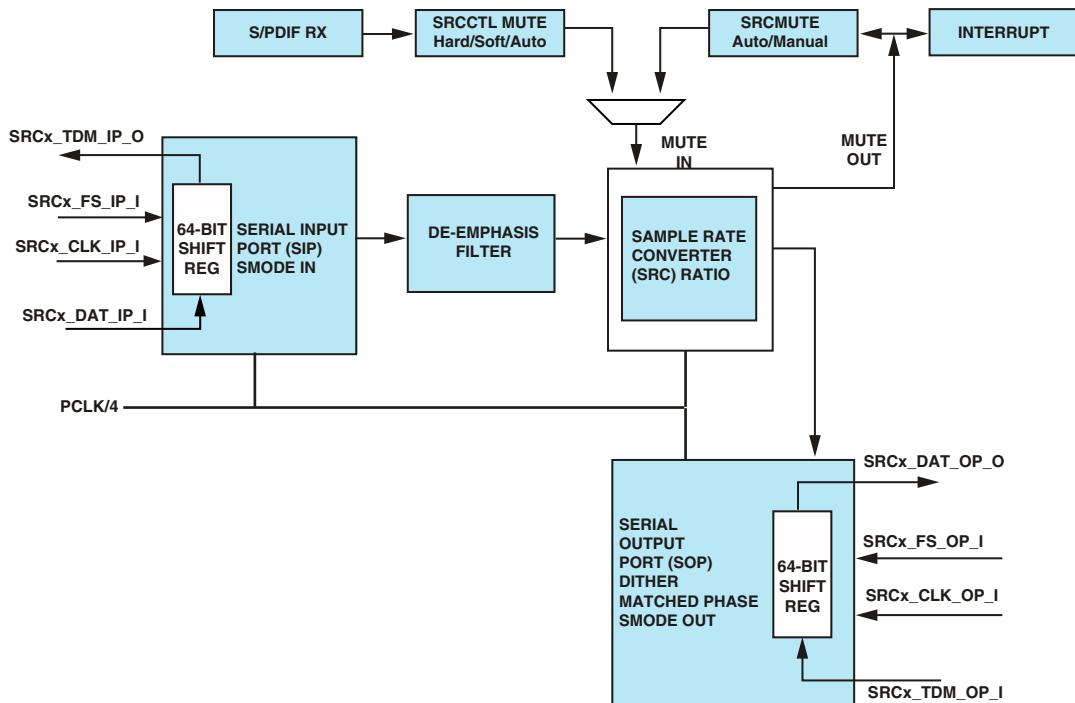


Figure 12-1. Sample Rate Converter Block Diagram

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 $(SRCx_FS_OP)/(SRCx_FS_IP)$ when $SRCx_FS_OP < SRCx_FS_IP$. The FIFO also scales the input data to mute and stop muting the SRC.

The RAM in the FIFO is 512 words deep for both left and right channels. An offset of 64 to the write address, provided by the $SRCx_FS_IP$ counter, is added to prevent the RAM read pointer from overlapping the write

Asynchronous Sample Rate Converter

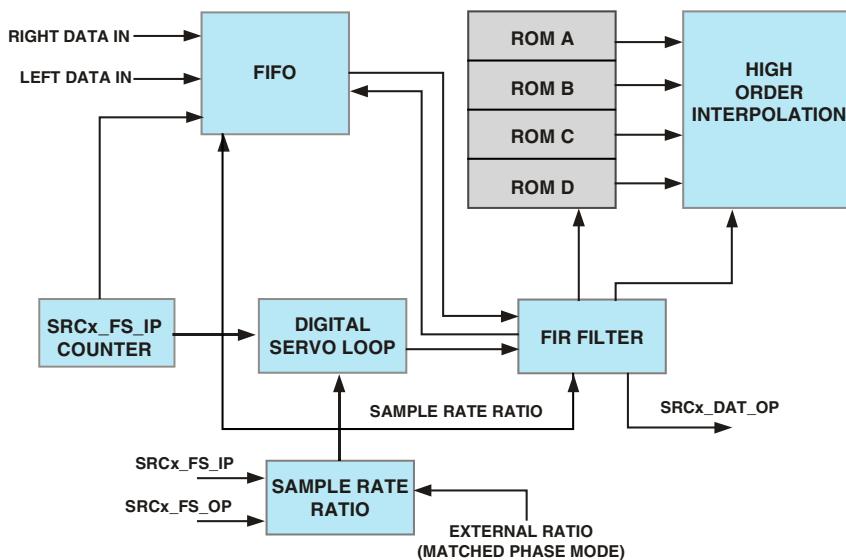


Figure 12-2. Sample Rate Converter Architecture

address. This offset value is useful for applications when small changes in the sample rate ratio between `SRCx_FS_IP` and `SRCx_FS_OP` are expected. The maximum decimation rate can be calculated from the RAM word depth is $(512 - 64) \div 64$ taps = 7.

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 `SRCx_FS_IP` and `SRCx_FS_OP` clocks as well as measure the arrival of the `SRCx_FS_OP` clock within 4.97 ps. The digital-servo loop also divides the fractional part of the ramp output by the ratio of $(SRCx_FS_IP)/(SRCx_FS_OP)$ for the case when $SRCx_FS_IP > SRCx_FS_OP$, to dynamically alter the ROM coefficients.

The digital-servo loop is implemented with a multi-rate filter. To settle the digital-servo loop filter quickly at startup or at a change in the sample

Functional Description

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 SRCx_MUTE_OUT bit 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 $\text{SRCx_FS_OP} < \text{SRCx_FS_IP}$ and is $(\text{SRCx_FS_IP})/(\text{SRCx_FS_OP}) \times 64$ taps for the case when $\text{SRCx_FS_IP} > \text{SRCx_FS_OP}$. The FIR filter performs its convolution by loading in the starting address of the RAM address pointer and the ROM address pointer from the digital-servo loop at the start of the SRCx_FS_OP period. The FIR filter then steps through the RAM by decrementing its address by 1 for each tap, and the ROM pointer increments its address by the $(\text{SRCx_FS_OP}/\text{SRCx_FS_IP}) \times 2^{20}$ ratio for $\text{SRCx_FS_IP} > \text{SRCx_FS_OP}$ or 2^{20} for $\text{SRCx_FS_OP} < \text{SRCx_FS_IP}$. Once the ROM address rolls over, the convolution is complete. The convolution is performed for both the left and right channels, and the multiply/accumulate circuit used for the convolution is shared between the channels.

The $(\text{SRCx_FS_IP})/(\text{SRCx_FS_OP})$ sample rate ratio circuit is used to dynamically alter the coefficients in the ROM for the case when $\text{SRCx_FS_IP} > \text{SRCx_FS_OP}$. The ratio is calculated by comparing the output of an SRCx_FS_OP counter to the output of an SRCx_FS_IP counter. If $\text{SRCx_FS_OP} > \text{SRCx_FS_IP}$, the ratio is held at one. If $\text{SRCx_FS_IP} > \text{SRCx_FS_OP}$, the sample rate ratio is updated if it is different by more than two SRCx_FS_OP periods from the previous SRCx_FS_OP to SRCx_FS_IP comparison. This is done to provide some hysteresis to prevent the filter length from oscillating and causing distortion.

However, the hysteresis of the $(\text{SRCx_FS_OP})/(\text{SRCx_FS_IP})$ 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 SRCx_FS_OP periods to update the SRCx_FS_OP and SRCx_FS_IP

ratios, two SRCs may have differences in their ratios from 0 to 4 SRC_x_FS_OP period counts. The (SRC_x_FS_OP)/(SRC_x_FS_IP) 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 SRC_x_FS_OP and SRC_x_FS_IP counters. The greater the resolution of the counters, the smaller the phase difference error.

Serial Data Ports

The serial data ports provide the interface through which data is transferred into and out of the SRC modules.

The SRC has a 3-wire interface for the serial input and output ports that supports left-justified, I²S, and right-justified (16-, 18-, 20-, 24-bit) modes. Additionally, the serial interfaces support 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.

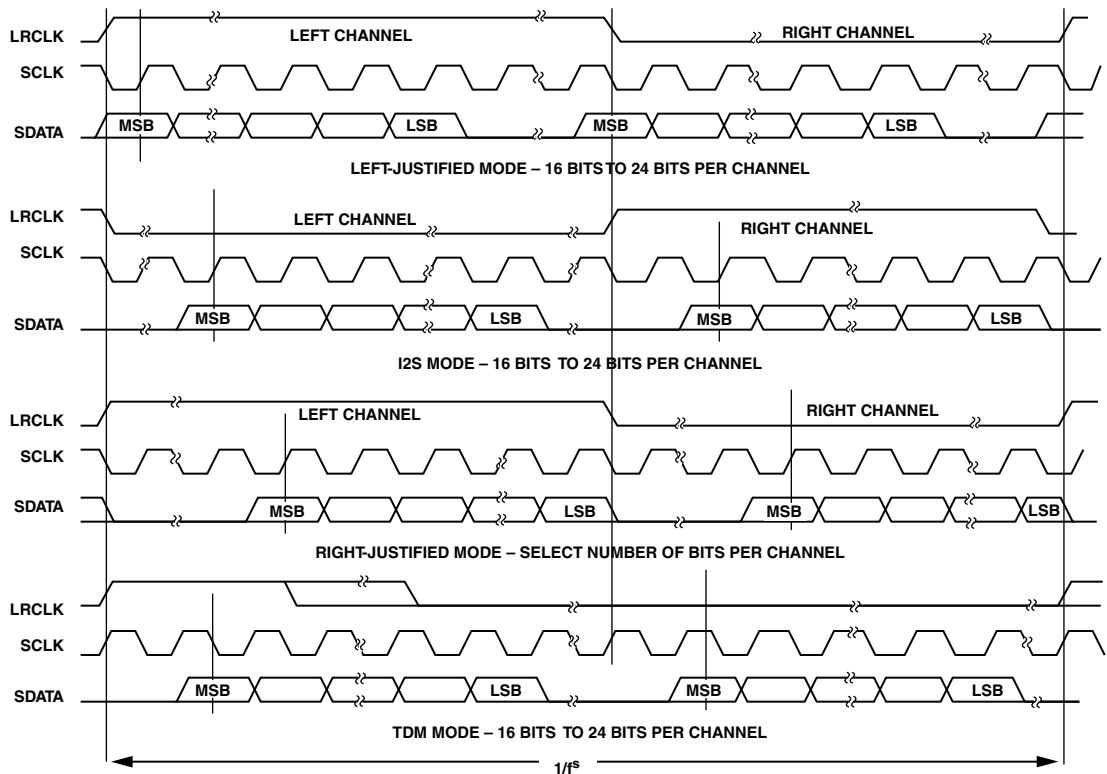
The SRC converts the data from the serial input port to the sample rate of the serial output port. The sample rate at the serial input port can be asynchronous with respect to the output sample rate of the output serial port.

Operating Modes

The SRC can operate in TDM, I²S, left-justified, right-justified, and bypass modes. The serial ports of the processor can be used for moving the SRC data to/from the internal memory.

In I²S, left-justified and right-justified modes, the SRCs operate individually. The serial data provided in the input port is converted to the sample rate of the output port. [Figure 12-3](#) shows the timing in the various formats.

Operating Modes



NOTES

¹ LRCLK NORMALLY OPERATES AT ASSOCIATIVE INPUT OR OUTPUT SAMPLE FREQUENCY (f_s).

² SCLK FREQUENCY IS NORMALLY $64 \times$ LRCLK EXCEPT FOR TDM MODE WHICH IS $N \times 64 \times f_s$, WHERE N = NUMBER OF STEREO CHANNELS IN THE TDM CHAIN.

³ PLEASE NOTE THAT 8 BITS OF EACH 32-BIT SUBFRAME ARE USED FOR TRANSMITTING MATCHED-PHASE MODE DATA.

Figure 12-3. SRC Data Format

TDM Daisy Chain Mode

The SRCs are daisy chained together to achieve the TDM mode of operation.

TDM Input Daisy Chain

In TDM input port, several SRCs can be daisy-chained together and connected to the serial input port of a SHARC processor or other processor (Figure 12-4). The SRC IP contains a 64-bit parallel load shift register. When the `SRCx_FS_IP_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 `SRCx_DATA_IP_I`, while the output is connected to `SRCx_TDM_IP_0`. By connecting the `SRCx_TDM_IP_0` to the `SRCx_DATA_IP_I` of the next SRC, a large shift register is created, which is clocked by `SRCx_CLK_IP_I`.



The number of SRCs that can be daisy-chained together is limited by the maximum frequency of `SRCx_CLK_xx_I`, refer datasheet for exact value. For example, if the maximum frequency of `SRCx_CLK_xx_I` is X MHz, and the output sample rate is f_S , then number of SRCs (n) that can be connected in daisy chained fashion is: $n \times 64 \times f_S \leq X$ MHz.

TDM Output Daisy Chain

In TDM output port, several SRCs can be daisy-chained together and connected to the SPORT of an ADSP-214xx or other processor (Figure 12-4). The SRC OP contains a 64-bit parallel load shift register. When the `SRCx_FS_OP_I` pulse arrives, each SRC loads its left and right data into the 64-bit shift register. The input to the shift register is connected to `SRCx_TDM_OP_I`, and the output is connected to `SRCx_DAT_OP_0`. By connecting the `SRCx_DAT_OP_0` to the `SRCx_TDM_OP_I` of the next SRC, a large shift register is created, which is clocked by `SRCx_CLK_OP_I`.

Operating Modes

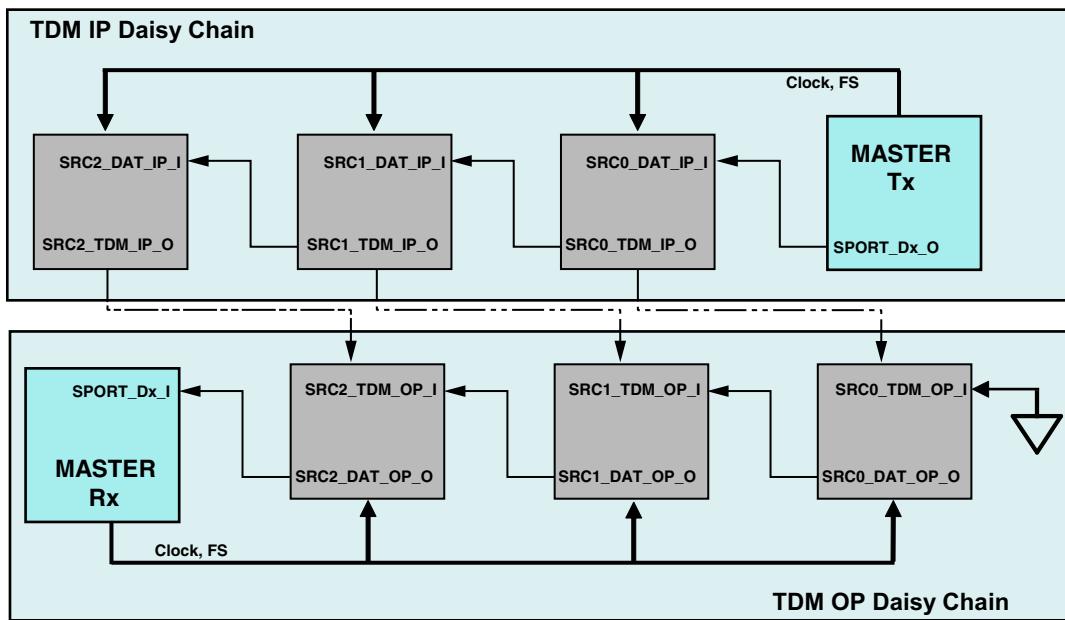


Figure 12-4. TDM Input/Output Modes

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 SRCx_FS_IP_I and SRCx_FS_OP_I 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.

Matched-Phase Mode (ADSP-21488)

The matched phase mode of the sample rate converter is enabled by the SRCx_MPPhase bit. This mode is used to match the phase (group delay)

between two or more adjacent sample rate converters that are operating with the same input and output clocks. When the SRC_x_MPHASE bit is set (=1), the SRC, a matched phase mode slave accepts the sample rate ratio transmitted by another SRC, the matched phase mode master, through its serial output as shown in [Figure 12-5](#).

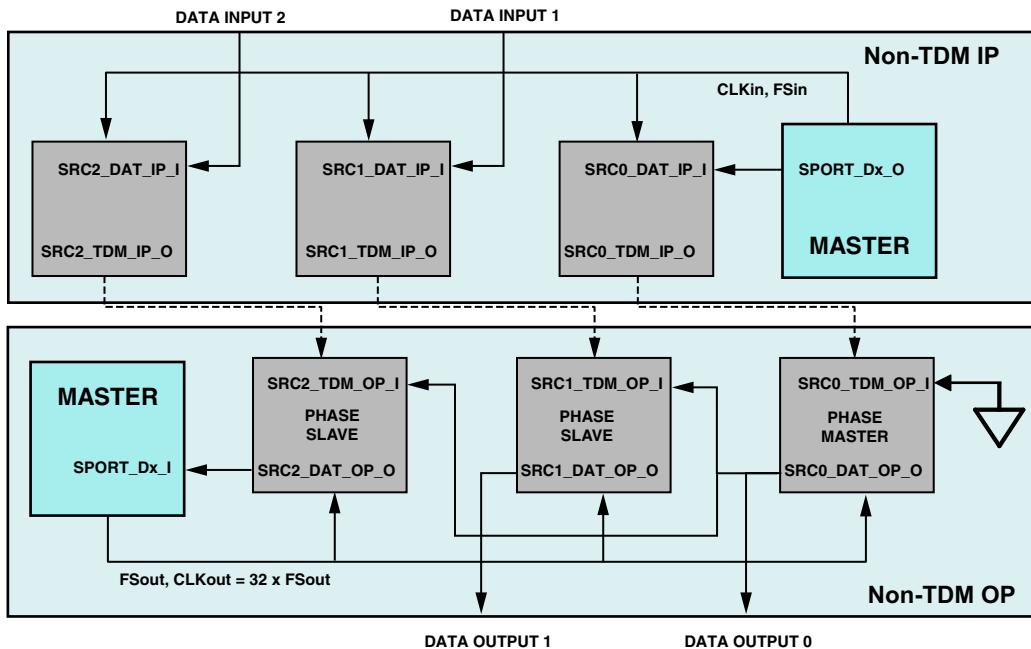


Figure 12-5. Typical Configuration for Matched-Phase Mode Operation

The phase master SRC device transmits its SRC_x_FS_OP/SRC_x_FS_IP ratio through the data output pin (SRC_x_DAT_OP_O) to the slave's SRC's data input pins (SRC_x_TDM_OP_I). The transmitted data (32-bit subframe) contains 24-bit data and 8-bits matched phase. The slave SRCs receive the 8-bit matched phase bits (instead of their own internally-derived ratio) if their SRC_x_MPHASE bits set to 1, respectively.

The SRC_x_FS_IP and SRC_x_FS_OP signals may be asynchronous with respect to each other in this mode. Note there must be 32 SRC_x_CLK_OP

Operating Modes

cycles per subframe in matched-phase mode (24-bits data and 8-bits phase match).

Data Format Matched-Phase Mode

The SRC supports the matched-phase mode for all serial output data formats; left-justified, I²S, right-justified, and TDM mode. Note that in the left-justified, I²S, and TDM modes, the lower 8 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 12-6](#).

AUDIO DATA LEFT CHANNEL, 24 BITS	MATCHED-PHASE DATA, 8 BITS	AUDIO DATA RIGHT CHANNEL, 24 BITS	MATCHED-PHASE DATA, 8 BITS
-------------------------------------	-------------------------------	--------------------------------------	-------------------------------

Left-Justified, I²S, and TDM Mode

MATCHED-PHASE DATA, 8 BITS	AUDIO DATA LEFT CHANNEL, 16 BITS - 24 BITS	MATCHED-PHASE DATA, 8 BITS	AUDIO DATA RIGHT CHANNEL, 16 BITS - 24 BITS
-------------------------------	---	-------------------------------	--

Right-Justified Mode

Figure 12-6. Matched-Phase Data Transmission

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 (phase mismatches) between multiple SRCs. The filter group delay of the SRC is given by the equations:

$$GDS = \frac{16}{SRCx_FS_IP} + \frac{32}{SRCx_FS_IP} \text{ seconds for } (SRCx_FS_OP \geq SRCx_FS_IP)$$

$$GDS = \frac{16}{SRCx_FS_IP} + \left(\frac{32}{SRCx_FS_IP} \right) \times \left(\frac{SRCx_FS_IP}{SRCx_FS_OP} \right) \text{ seconds for } (SRCx_FS_OP \leq SRCx_FS_IP)$$

Decimation Rate

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 ever overlapping the write address. The offset is fixed by the group delay signal. A small offset, 16, is added to the write address pointer.

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 = 7.75:1.

Muting Modes

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. Automatic muting can be disabled by setting (=1) the SRCx_MUTE_EN bits in the SRCMUTE register.



Note that by default, the SRCMUTE register connects the MUTE_IN signal to the MUTE_OUT signal, but not vice versa.

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.

Interrupts

A 12-bit counter, clocked by `SRCx_FS_IP_I`, is used to control the mute attenuation. Therefore, the time it takes from the assertion of `MUTE_IN` to -144 dB, full mute attenuation is 4096 FS cycles.

Likewise, the time it takes to reach 0 dB mute attenuation from the deassertion of `MUTE_IN` is 4096 FS cycles.

Hard Mute

When the `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 `SRCCTLx` register is set, the SRC communicates with the S/PDIF receiver peripheral to determine when the input should mute. Each SRC is connected to the S/PDIF receiver to read the `DIR_NOAUDIO` bits. When the `DIR_NOAUDIO` bit is set (=1), 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.

Interrupts

The SRC mute-out signal can be used to generate interrupts on their rising edge, falling edge, or both, depending on how the DAI interrupt mask registers (`DAI_IMASK_RE/FE`) are programmed. This allows the generation of `DAIHI/DAILI` interrupts either entering mute, exiting muting or both.

The `SRCx_MUTE_OUT` interrupt is generated only once when the SRC is locked (after 4096 FS input samples) and after changes to the sample

ratio. Hard mute, soft mute, and auto mute only control the muting of the input data to the SRC.

[Table 12-4](#) provides an overview of SRC interrupts.

Table 12-4. SRC Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
DAI SRC RX/TX (I2S, left/right justified, TDM, 4 channels)	– Mute out asserted (SRC init, SRC sample rate change)		Read-to-clear DAI_IRPTL_x + RTI instruction	P0I, P12I

Debug Features

The asynchronous sample rate converter allow the bypass mode. When the BYPASS bit is set (=1), the input data bypasses the sample rate converter and is sent directly to the serial output port. This mode can be used for testing both ports when the input and output sample rates are at the same frequency, therefore both in- and output ports can be routed to the same serial clock and frame sync.

Effect Latency

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

Write Effect Latency

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

Effect Latency

SRC Effect Latency

After the ASRC registers are configured the effect latency is 1.5 PCLK cycles minimum and 3 PCLK cycles maximum.

13 SONY/PHILIPS DIGITAL INTERFACE

The Sony/Philips Digital Interface (S/PDIF) is a standard audio data transfer format that allows the transfer of digital audio signals from one device to another without having to convert them to an analog signal. The digital audio interface carries three types of information; audio data, non audio data (compressed data) and timing information. Its specifications are listed in [Table 13-1](#).

Table 13-1. S/PDIF Specifications

Feature	Transmitter	Receiver
Connectivity		
Multiplexed Pinout	No	No
SRU DAI Required	Yes	Yes
SRU DAI Default Routing	No	No
SRU2 DPI Required	No	No
SRU2 DPI Default Routing	N/A	N/A
Interrupt Control	Yes	Yes
Protocol		
Master Capable	No	Yes
Slave Capable	Yes	No
Transmission Simplex	Yes	Yes
Transmission Half Duplex	No	No
Transmission Full Duplex	No	No

Features

Table 13-1. S/PDIF Specifications (Cont'd)

Feature	Transmitter	Receiver
Access Type		
Data Buffer	No	No
Core Data Access	N/A	N/A
DMA Data Access	N/A	N/A
DMA Channels	N/A	N/A
DMA Chaining	N/A	N/A
Boot Capable	N/A	N/A
Local Memory	No	No
Clock Operation	$f_{PCLK}/4$	$f_{PCLK}/4$

Features

The S/PDIF interface has the following features.

- AES3-compliant S/PDIF transmitter and receiver.
- Transmitting a biphase mark encoded signal that may contain any number of audio channels (compressed or linear PCM) or non-audio data.
- S/PDIF receiver managing clock recovery with separate S/PDIF PLL or optional using external PLL circuit.
- S/PDIF receiver direct supports DTS frames of 256, 512, 1024, 2048, and 4096 (4096 frames are not supported for the ADSP-2146x processors).
- Managing user status information and providing error-handling capabilities in both the transmitter and receiver.

- DAI allows interactions over DAI by serial ports, IDP and/or the external DAI pins to interface to other S/PDIF devices. This includes using the receiver to decode incoming biphase encoded audio streams and passing them via the SPORTs to internal memory for processing-or using the transmitter to encode audio or digital data and transfer it to another S/PDIF receiver in the audio system.

Notice it is important to be familiar with serial digital audio interface standards IEC-60958, EIAJ CP-340, AES3 and AES11.

Pin Descriptions

[Table 13-2](#) provides descriptions of the pins used for the S/PDIF transmitter.

Table 13-2. S/PDIF Transmitter Pin Descriptions

Internal Node	I/O	Description
DIT_CLK_I	Input	Serial clock. Controls the rate at which serial data enters the S/PDIF module. This is typically 64 time slots. ¹
DIT_DAT_I	Input	Serial Data. The format of the serial data can be I ² S, and right- or left-justified.
DIT_FS_I	Input	Serial Frame Sync.
DIT_HFCLK_I	Input	Input sampling clock. The over sampling clock (which is divided down according to the FREQMULT bit in the transmitter control register to generate the biphase clock)
DIT_EXTSYNC_I	Input	External Synchronization. Used for synchronizing the frame counter. If External synchronization is enabled (bit 15 of DITCTL is set), Frame counter resets at rising edge of LRCLK next to the rising edge of EXT_SYNC_I.

SRU Programming

Table 13-2. S/PDIF Transmitter Pin Descriptions (Cont'd)

Internal Node	I/O	Description
DIT_O	Output	Transmit Biphase Mark Encoded Data Stream.
DIT_BLKSTART_O	Output	Transmit Block Start. Indicates the last frame of the current block. This is high for the entire duration of the last frame. This can also be connected to the DAI interrupts 31–22 using SRU_MISCx registers.

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 in the “[Features](#)” section of this chapter.

[Table 13-3](#) provides descriptions of the pins used for the S/PDIF receiver.

Table 13-3. S/PDIF Receiver Pin Descriptions

Internal Node	I/O	Description
SPDIF_EXTPLLCLK_I	Input	PLL clock input ($512 \times FS$). Input clock from external PLL.
DIR_I	Input	Biphase mark encoded data receiver input stream.
DIR_CLK_O	Output	Extracted receiver sample clock output.
DIR_TDMCLK_O	Output	Receiver TDM clock out. This clock is $256 \times DIR_FS_O$.
DIR_FS_O	Output	Extracted receiver frame sync out.
DIR_DAT_O	Output	Extracted audio data output.
DIR_LRCLK_FB_O	Output	Receiver frame sync feed back output. Input for external PLL.
DIR_LRCLK_REF_O	Output	Receiver frame sync reference clock output. Input for external PLL.

SRU Programming

The SRU (signal routing unit) is used to connect the S/PDIF transmitter biphase data out to the output pins or to the S/PDIF receiver. The serial

clock, frame sync, data, and EXT_SYNC (if external synchronization is required) inputs also need to be routed through SRU (see [Table 13-4](#)).

Table 13-4. S/PDIF DAI/SRU Transmitter Signal Connections

Internal Node	DAI Group	SRU Register
Inputs		
DIT_CLK_I	Group A	SRU_CLK4-2
DIT_HFCLK_I		
DIT_EXTSYNC_I		
DIT_DAT_I	Group B	SRU_DAT4
DIT_FS_I	Group C	SRU_FS2
Outputs		
DIT_O	Group B, D	
DIT_BLKSTART_O	Group D, E	

The SRU (signal routing unit) needs to be programmed in order to connect the S/PDIF receiver to the output pins or any other peripherals and also for the connection to the input biphase stream.

Program the corresponding SRU registers to connect the outputs to the required destinations ([Table 13-5](#)). The biphase encoded data and the external PLL clock inputs to the receiver are routed through the signal routing unit (SRU). The extracted clock, frame sync, and data are also routed through the SRU.

Table 13-5. S/PDIF DAI/SRU Receiver Signal Connections

Internal Node	DAI Group	SRU Register
Inputs		
SPDIF_EXTPLLCLK_I	Group A	SRU_CLK4
DIR_I	Group C	SRU_DAT5

Register Overview

Table 13-5. S/PDIF DAI/SRU Receiver Signal Connections (Cont'd)

Internal Node	DAI Group	SRU Register
Outputs		
DIR_CLK_O	Group A, D	
DIR_TDMCLK_O		
DIR_FS_O	Group C, D	
DIR_DAT_O	Group B, D	
DIR_LRCLK_FB_O	Group D	
DIR_LRCLK_REF_O		

Register Overview

This section provides brief descriptions of the major registers. For complete information see “[Sony/Philips Digital Interface Registers](#)” on page A-199.

Transmit Control Register (DITCTL). The `DITCTL` register contains control parameters for the S/PDIF transmitter. The control parameters include transmitter enable, mute information, over sampling clock division ratio, SCDF mode select and enable, serial data input format select and validity and channel status buffer selects.

Transmit Channel Status Registers (DITCHANAx/Bx). These registers provide status bit information for transmitter subframe A and B in stand-alone mode.

Transmit User Bit Registers (DITUSRBITAx/Bx). These registers provide user bit information for transmitter subframe A and B in standalone mode.

Receive Control Register (DIRCTL). 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 S/PDIF PLL disable.

Receive Status Register (DIRSTAT). The receiver also detects errors in the S/PDIF stream. These error bits are stored in the status register, which can be read by the core. Optionally, an interrupt may be generated to notify the core on error conditions.

Receive Channel Status Registers (DIRCHANAx/Bx). These registers provide status information for receiver subframe A and B.

Clocking

The fundamental timing clock of the S/PDIF is peripheral clock/4 (PCLK/4).

S/PDIF Transmitter

The following sections provide information on the S/PDIF transmitter.

Functional Description

The S/PDIF transmitter, shown in [Figure 13-1](#) resides within the DAI, and its inputs and outputs can be routed via the SRU. It receives audio data in serial format, encloses the specified user status information, and converts it into the biphase encoded signal. The serial data input to the transmitter can be formatted as left-justified, I²S, or right-justified with word widths of 16, 18, 20 or 24 bits. [Figure 13-2](#) shows detail of the AESs block.

The serial data, clock, and frame sync inputs to the S/PDIF transmitter are routed through the signal routing unit (SRU). [For more information, see “DAI Signal Routing Unit Registers” on page A-118.](#)

The S/PDIF transmitter output may be routed to an output pin via the SRU and then routed to another S/PDIF receiver or to components for

S/PDIF Transmitter

off-board connections to other S/PDIF receivers. The output is also available to the S/PDIF receiver for loop-back testing through SRU.

In addition to encoding the audio data in the bi-phase format, the transmitter also provides a way to easily add the channel status information to the outgoing bi-phase stream. There are status/user registers for a frame (192-bits/24 bytes) in the transmitter that correspond to each channel or subframe. For more information, see “[Transmitter Registers](#)” on [page A-199](#).

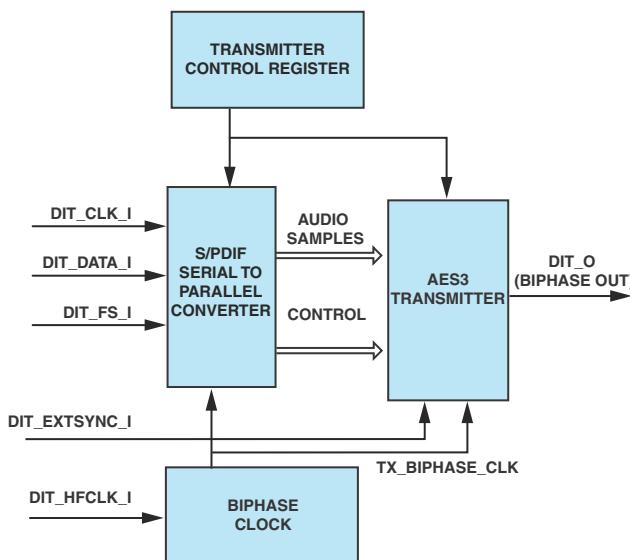


Figure 13-1. S/PDIF Transmitter Block Diagram

Validity bits for both channels may also be controlled by the transmitter control register. Optionally, the user bit, validity bit, and channel status bit are sent to the transmitter with each left/right sample. For each subframe the parity bit is automatically generated and inserted into the bi-phase encoded data. A mute control and support for double-frequency single-channel mode are also provided. The serial data input format may be selected as left-justified, I²S, or right-justified with 16-, 18-, 20- or

24-bit word widths. The over sampling clock is also selected by the transmitter control register.

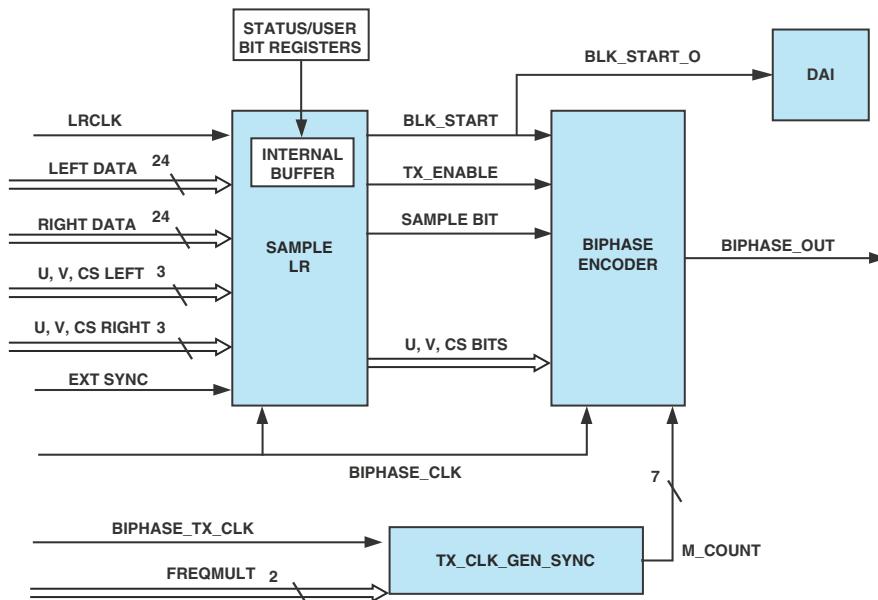


Figure 13-2. AES3 Output Block

Input Data Format

The Figure 13-3 through Figure 13-7 shows the format of data that is sent to the S/PDIF transmitter using a variety of interfaces.

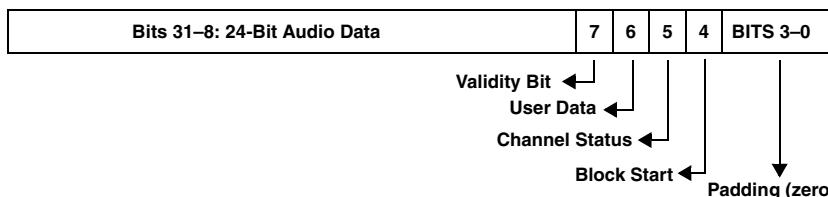


Figure 13-3. Data Packing for I²S and Left-Justified Format

S/PDIF Transmitter



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.

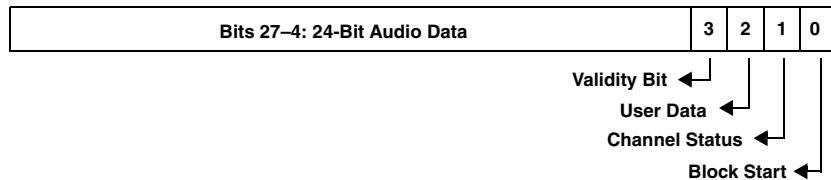


Figure 13-4. Data Packing for Right-Justified Format, 24 Bits

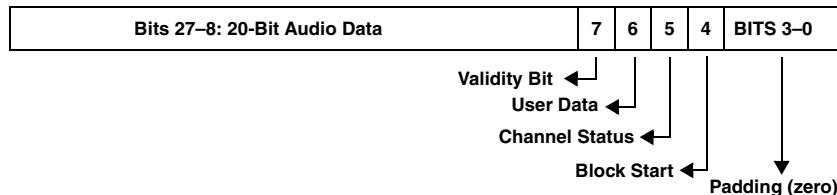


Figure 13-5. Data Packing for Right-Justified Format, 20 Bits

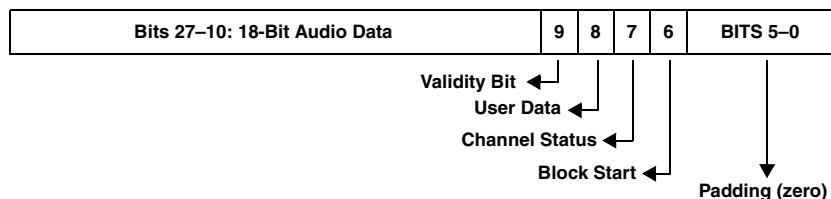


Figure 13-6. Data Packing for Right-Justified Format, 18 Bits

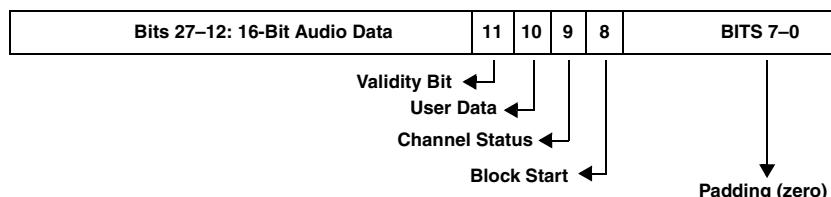


Figure 13-7. Data Packing for Right-Justified Format, 16 Bits

Operating Modes

The S/PDIF transmitter can operate in standalone and full serial modes. The following sections describe these modes in detail.

Full Serial Mode

This mode is selected by clearing bit 9 in the `DITCTL` register. In this mode all the status bits, audio data and the block start bit (indicating start of a frame), come through the serial data stream (`DIT_DATA_I`) pin. The transmitter should be enabled after or at the same time as all of the other control bits.

Standalone Mode

This mode is selected by setting bit 9 in the `DITCTL` register. In this mode, the block start bit (indicating the start of a frame) is generated internally. The channel status bits come from the channel status buffer registers (`DITCHANAX` and `DITCHANBX`). The user status bits come from the user bits buffers (`DITUSRBITAX` and `DITUSRBITBX`) as shown in [Figure 13-2 on page 13-9](#).



The channel status buffer must be programmed before the S/PDIF transmitter is enabled and used for all the successive blocks of data.

The validity bit for channel A and B are taken from bit 10 and bit 11 of the `DITCTL` register. In this mode only audio data comes from the `DIT_DATA_I` pin. All other data, including the status bit and block start bit is either generated internally or taken from the internal register.

Once the user bits buffer registers (`DITUSRBITA0-5` and `DITUSRBITB0-5`) are programmed, they are used only for the next block of data. This allows programs to change the user bit information in every block of data.

S/PDIF Transmitter

To allow user bit updates, write a 0x1 to the `DIT_USRUPD` register that is used for further processing. If the `DIT_AUTO` bit in the `DITCTL` register is set:

- At every 192nd Frame end, if `DITUSRUPD` = 1, then the user status bits are taken from user bits buffers and transmitted. Simultaneously, the `DIT_USRUPD` register is cleared automatically by hardware.
- At every 192nd Frame end, if `DITUSRUPD` = 0 then the user status bits are updated as zeros and transmitted. The `DIT_USRUPD` register remains low.



For the first block of transfer, write a one (1) to the `DITUSRUPD` register and then enable the S/PDIF transmitter.

In general, for the next block, programs can update user bits buffers at any time during the transfer of the current block (1 block = 192 frames).

There are internal buffers to store the user status bits of the current block of transfer. In other words, at the beginning of every new block, the user status bit (`DIT_USRPEND` in the `DITCTL` register) from user bits buffers are copied to internal buffers and transmitted in each frame during the transfer.

Note that since a frame contains $192 \text{ bits}/8 = 24 \text{ bytes}$, six status/user registers are required to store each four bytes.

Data Output Mode

Two output data formats are supported by the transmitter; *two channel mode* and *single-channel double-frequency* (SCDF) mode. The output format is determined by the transmitter control register (`DITCTL`).

In two channel mode, the left channel (channel A) is transmitted when the `DIT_FS_I` is high and the right channel (channel B) is transmitted when the `DIT_FS_I` is low.

In SCDF mode, the transmitter sends successive audio samples of the same signal across both sub frames, instead of channel A and B. The transmitter will transmit at half the sample rate of the input bit stream. The DIT_SCDF bit (bit 4 in the DITCTL register) selects SCDF mode. When in SCDF mode, the DIT_SCDF_LR bit (bit 5 in the DITCTL register) register decides whether left or right channel data is transmitted.

S/PDIF Receiver

The S/PDIF receiver (Figure 13-8) is compliant with all common serial digital audio interface standards including IEC-60958, IEC-61937, AES3, and AES11. These standards define a group of protocols that are commonly associated with the S/PDIF interface standard defined by AES3, which was developed and is maintained by the Audio Engineering Society. The AES3 standard effectively defines the data and status bit structure of an S/PDIF stream. AES3-compliant data is sometimes referred to as AES/EBU compliant. This term highlights the adoption of the AES3 standard by the European Broadcasting Union.

Functional Description



The S/PDIF receiver is enabled at default to receive in two-channel mode. If the receiver is not used, programs should disable the receiver as the digital PLL may produce unwanted switching noise.

If the receiver is not used, programs should disable the digital PLL to avoid unnecessary switching. This is accomplished by writing into the DIR_RESET bit in the DIRCTL register. In most cases, when the S/PDIF receiver is used, this register does not need to be changed. After the SRU programming is complete, write to the DIRCTL register with control values. At this point, the receiver attempts to lock.

For a detailed description of this register, see “[Receive Control Register \(DIRCTL\)](#)” on page [A-204](#).

S/PDIF Receiver

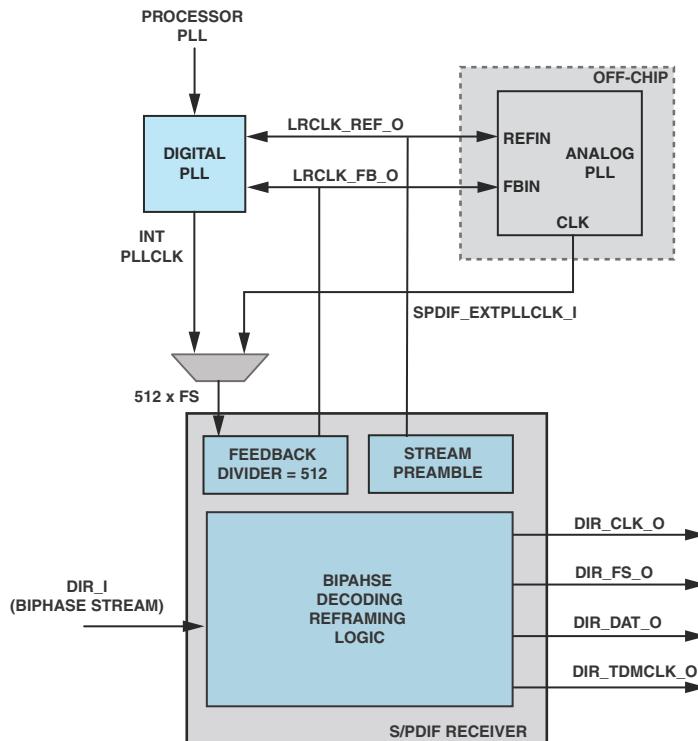


Figure 13-8. S/PDIF Receiver Block Diagram

The input to the receiver (`DIR_I`) is a biphase encoded signal that may contain two audio channels (compressed or linear PCM) or non-audio data. The receiver decodes the single biphase encoded stream, producing an I²S compatible serial data output that consists of a serial clock, a left-right frame sync, and data (channel A/B). It provides the programmer with several methods of managing the incoming status bit information.

The S/PDIF receiver receives any S/PDIF stream with a sampling frequency range of 32 kHz – 15% to 192 kHz + 15% range.

The channel status bits are collected into memory-mapped registers, while other channel status and user bytes must be handled manually. The block

start bit, which replaces the parity bit in the serial I²S stream, indicates the reception of the Z preamble and the start of a new block of channel status and data bits.

Clock Recovery

The phased-locked loop for the AES3/SPDIF receiver is intended to recover the clock that generated the AES3/SPDIF biphase encoded stream. This clock is used by the receiver to clock in the biphase encoded data stream and also to provide clocks for either the SPORTs, sample rate converter, or the AES3/SPDIF transmitter. The recovered clock may also be used externally to the chip for clocking D/A and A/D converters.

In order to maintain performance, jitter on the clock is sourced to several peripherals. 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 ±15% in frequency over time. This allows for applications that do not use PLL unlocking.

To be AES11 compliant, the recovered left/right clock must be aligned with the preambles within a + or – 5% of the frame period. Since the PLL generates a clock 512 times the frame rate clock ($512 \times \text{FSCLK}$), this clock can be used and divided down to create the phase aligned jitter-free left/right clock. For more information on recovered clocks, see “[Clock Recovery](#)” on page 13-15.

Output Data Format

The extracted 24-bit audio data, V, U, C and block start bits are sent on the DIR_DAT_0 pin in 32-bit I²S format as shown in [Figure 13-3](#). The frame sync is transmitted on the DIR_FS_0 pin and serial clock is transmitted on the DIR_CLK_0 pin. All three pins are routed through the SRU.

Channel Status

The channel status for the first bytes 4–0 (consumer mode) are collected into memory-mapped registers (`DIRCTL` and `DIRCHANA/DIRCHANB` registers). All other channel status bytes 23–5 (professional mode) must be manually extracted from the receiver data stream.



Only the first 5 channel status bytes (40-bit) for consumer mode of a frame are stored into the S/PDIF receiver status registers.

Operating Modes

This section describes the receiver channel status for the different modes.

Compressed or Non-linear Audio Data

The S/PDIF receiver processes compressed as well as non-linear audio data according to the supported standards. The following sections describe how this peripheral handles different data.

The AES3/SPDIF receiver is required to detect compressed or non-linear audio data according to the AES3, IEC60958, and IEC61937 standards. Bit 1 of byte 0 in the `DIRSTAT` register indicates whether the audio data is linear PCM, (bit 1=0), or non-PCM audio, (bit 1=1). If the channel status indicates non-PCM audio, the `DIR_NOAUDIO` bit flag is set. (This bit can be used to generate an interrupt.) The `DIR_VALID` bit (bit 3 in the `DIRSTAT` register) when set (=1) may indicate non-linear audio data as well. Whenever this bit is set, the `VALIDITY` bit flag is set in the `DIR_RX_STAT` register.

MPEG-2, AC-3, DTS, and AAC compressed data may be transmitted without setting either the `DIR_VALID` bit or bit 1 of byte 0. To detect this data, the IEC61937 and SPMTE 337M standards dictate that there be a 96-bit sync code in the 16-, 20- or 24-bit audio data stream. This sync code consists of four words of zeros followed by a word consisting of `0xF872` and another word consisting of `0x4E1F`. When this sync code is

detected, the `DIR_NOAUDIO` bit flag is set. If the sync code is not detected again within 4096 frames, the `DIR_NOAUDIO` bit flag is deasserted.

The last two words of the sync code, 0xF872 and 0x4E1F, are called the preamble-A and preamble-B of the burst preamble. Preamble-C of the burst preamble contains burst information and is captured and stored by the receiver. Preamble-D of the burst preamble contains the length code and is captured by the receiver. Even if the validity bit or bit 1 of byte 0 has been set, the receiver still looks for the sync code in order to record the preamble-C and D values. Once the sync code has not been detected in 4096 frames, the preamble-C and D registers are set to zero.



The S/PDIF receiver in 2147x and 2148x, supports all DTS frame sizes of 256, 512, 1024, 2048 and 4096. To enable support for 2048 and 4096 DTS frame sizes, `DTS_CD_4K_EN` bit in `DIRCTL`, need to be set. In 2146x on-chip S/PDIF receiver supports 256, 512 and 1024 DTS frames only. The DTS test kit frames with 2048 and 4096 frame sizes can be detected by adding the sync detection logic in software by using a software counter to check for the DTS header every 2048 and 4096 frames respectively.

Emphasized Audio Data

The receiver must indicate to the program whether the received audio data is emphasized using the channel status bits as detailed below.

- In professional mode, (bit 0 of byte 0 = 1), channel status bits 2–4 of byte 0 indicate the audio data is emphasized if they are equal to 110 or 111.
- In consumer mode, (bit 0 of byte 0 = 0), channel status bits 3–5 indicate the audio data is emphasized if they are equal to 100, 010 or 110.

If emphasis is indicated in the channel status bits, the receiver asserts the `EMPHASIS` bit flag. This bit flag is used to generate an interrupt.

Single-Channel Double-Frequency Mode

Single-channel, double-frequency mode (SCDF) mode is selected with DIR_SCDF and DIR_SCDF_LR bits in the DIRCTL register. The DIR_BOCHANL/R bits in the DIRSTAT register also contain information about the SCDF mode. When the DIR_BOCHANL/R indicates single channel double frequency mode, the two subframes of a frame carry successive audio samples of the same signal. Bits 0–3 of channel status byte 1 are decoded by the receiver to determine one of the following:

- 0111 = single channel double frequency mode
- 1000 = single channel double frequency mode—stereo left
- 1001 = single channel double frequency mode—stereo right

Clock Recovery Modes

The S/PDIF receiver extracts audio data, channel status, and user bits from the biphase encoded AES3 and S/PDIF stream. In addition, a 50% duty cycle reference clock running at the sampling rate of the audio input data is generated for the PLL in the receiver to recover the oversampling clock.

Digital On-Chip PLL

The receiver can recover the clock from the biphase encoded stream using an on-chip digital PLL shown in [Figure 13-8](#). Note the dedicated on-chip digital PLL is separate from the PLL that supplies the clock to the SHARC processor core and which is the default operation of the receiver.

The left/right frame reference clock for the PLL is generated using the preambles. The recovered low jitter left/right frame clock from the PLL attempts to align with the reference clock. However, this recovered left/right clock, like the reference clock, is not phase aligned with the preambles.

External Analog PLL

Notice there are various performance characteristics to consider when configuring for analog PLL mode. In order to provide the receiver with an external PLL the appropriate routings needs to be performed including the setting of `DIR_PLLDIS` bit which disables the internal PLL and connects to the external PLL. For more information about using PLLs, visit the Analog Devices Inc. web site at: <http://www.analog.com/en/clock-and-timing/pll-synthesizersvcos/products/index.html>

Interrupts

All S/PDIF interrupts are generated by the transmitter and receiver and processed through the DAI interrupt controller which can generate an interrupt signal using the (`DAI_IMASK_X`) registers.

[Table 13-6](#) provides an overview of S/PDIF interrupts.

Table 13-6. S/PDIF Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
DAI S/PDIF RX/TX (I2S, left/right justified, TDM, 2 channels)	<ul style="list-style-type: none"> – TX block start – RX audio status (no audio, status change, emphasis) – RX error (lock, validity, no stream, biphase, parity, CRC) 		Read-to-clear <code>DAI_IRPTL_X</code> + RTI instruction	P0I, P12I (S/PDIF RX only)

Transmitter Interrupt

The `DIT_BLKSTART_0` output signal, if routed to any miscellaneous interrupt bits (`DAI_INT_31-22` in the `SRU_MISCx` register), triggers a block start interrupt during the last frame of current block.

Receiver Interrupts

The following three receiver status bits (DAI_IRPTL_x) generate an interrupt.

- No audio (DIR_NOAUDIO_INT)
- Emphasized audio (DIR_EMPHASIS_INT)
- Status change (DIR_STATCNG_INT)

Note the Status change interrupt is generated if any of the 40 status bits (bytes 4–0) have changed.

Receiver Error Interrupts

The following five receiver error status bits (DAI_IRPTL_x) generate an interrupt.

- Receiver Locked (DIR_LOCK_INT)
- Validity (DIR_VALID_INT)
- No Audio Stream (DIR_NOSTREAM_INT)
- CRC Error (DIR_CRCERROR_INT)
- Parity or biphase Error (DIR_ERROR_INT)

Notice that parity error and biphase error are ORed together to form a DIR_ERROR_INT interrupt. The CRCCERROR bit is not available in DIRSTAT register. The CRCCERROR interrupt latch bit is set whenever the CRCC check of the channel status bits fails. The CRCC check is only performed if channel status bit 0 of byte 0 is high, indicating professional mode.

Debug Features

The following feature supports S/PDIF debugging.

Loop Back Routing

The S/PDIF supports an internal loopback mode by using the SRU. For more information, see “[Loop Back Routing](#)” on page 9-40.

Effect Latency

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

Write Effect Latency

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

Programming Model

The following sections provide information on programming the transmitter and receiver.

Programming the Transmitter

Since the S/PDIF transmitter data input is not available to the core, programming the transmitter is as simple as: 1) connecting the SRU to the on-chip (serial ports or input data port) or off-chip (DAI pins) serial devices that provide the clock and data to be encoded, and 2) selecting the

Programming Model

desired mode in the transmitter control register. This setup can be accomplished in three steps.

1. Connect the transmitter's four required input signals and one biphase encoded output in the SRU. The four input signals are the serial clock (`DIT_CLK_I`), the serial frame sync (`DIT_FS_I`), the serial data (`DIT_DAT_I`), and the high frequency clock (`DIT_HFCLK_I`) used for the encoding. The only output of the transmitter is `DIT_O`.
2. If user bits are required, write 0x1 to the `DITUSRUPD` register for the first block of transfer. Also route the `DIT_BLK_START_0` signal to the `DAI_INT_26` (`DAI_IRPTLx` register). This generates interrupts during the last frame of the block (192), allowing changes of user bits for the next block.
3. Initialize the `DITCTL` register to enable the data encoding.
4. Manually set the block start bit in the data stream once per block (384 words). This is necessary if automatic generation of block start information is not enabled in the `DITCTL` register, (`DIT_AUTO = 0`).

Programming the Receiver

Since the S/PDIF receiver data output is not available to the core, programming the peripheral is as simple as connecting the SRU to the on-chip (serial ports or input data port) or off-chip (DAI pins) serial devices that provide the clock and data to be decoded, and selecting the desired mode in the receiver control register. This setup can be accomplished in two steps.

1. Connect the input signal and three output signals in the SRU for. The only input of the receiver is the biphase encoded stream, `DIR_I`. The three required output signals are the serial clock

(DIR_CLK_0), the serial frame sync (DIR_FS_0), and the serial data (DIR_DAT_0). The high frequency clock (DIR_TDMCLK_0) derived from the encoded stream is also available if the system requires it.

2. Initialize the DIRCTL register to enable the data decoding. Note that this peripheral is enabled by default.

Interrupted Data Streams on the Receiver

When using the S/PDIF receiver with data streams that are likely to be interrupted, (in other words unplugged and reconnected), it is necessary to take some extra steps to ensure that the S/PDIF receiver's digital PLL will re lock to the stream. The steps to accomplish this are described below.

1. Set up interrupts within the DAI so that the S/PDIF receiver can generate an interrupt when the stream is reconnected.
2. Within the interrupt service routine (ISR), stop and restart the digital PLL. This is accomplished by setting and then clearing bit 7 of the S/PDIF receiver control register.
3. Return from the ISR and continue normal operation.

This method of resetting the digital PLL has been shown to provide extremely reliable performance when S/PDIF inputs that are interrupted or unplugged momentarily occur.

The following procedure and the example code show how to reset the digital PLL. Note that all of the S/PDIF receiver interrupts are handled through the DAI interrupt controller.

1. Initialize the No Stream Interrupt

```
/* Enable interrupts (globally) */  
BIT SET MODE1 IRPTEN;  
/* unmask DAI Hi=Priority Interrupt */
```

Programming Model

```
bit set imask DAIHI;
ustat1 = DIR_NOSTREAM_INT;

/* Enable no-stream Interrupt on Falling Edge. Interrupt
occurs when the stream is reconnected */
dm(DAI_IRPTL_FE) = ustat1;

/* Enable Hi-priority DAI interrupt */
dm(DAI_IRPTL_PRI) = ustat1;

/* If more than 1 DAI interrupt is being used, it is neces-
sary to determine which interrupt occurred here */

/* Interrupt Service Routine for the DAI Hi-Priority Inter-
rupt. This ISR triggered when the DIR sets no_stream bit */
_DAIisrH:
```

2. Reset the Digital PLL Inside of the ISR

```
r8=dm(DAI_IRPTL_H);           /* Reading DAI_IRPTL_H
                                clears interrupt */
ustat2=dm(DIRCTL);
bit set ustat2 DIR_PLLDIS; /* bit_7 disables Dpll only */
dm(DIRCTL)=ustat2;
bit clr ustat2 DIR_PLLDIS; /*reenable the digital pll */
dm(DIRCTL)=ustat2;
```

14 PRECISION CLOCK GENERATOR

The precision clock generators (PCG) consist of four units, each of which generates a pair of signals (clock and frame sync) derived from a clock input signal. The units, A B, C, and D, are identical in functionality and operate independently of each other. The two signals generated by each unit are normally used as a serial bit clock/frame sync pair. [Table 14-1](#) lists the PCG specifications.

Table 14-1. PCG Specifications

Feature	PCGA-B	PCGC-D
Connectivity		
Multiplexed Pinout	No	No
SRU DAI Required	Yes	Yes
SRU DAI Default Routing	No	No
SRU2 DPI Required	No	Yes
SRU2 DPI Default Routing	No	No
Interrupt Control	No	No
Protocol		
Master Capable	Yes	Yes
Slave Capable	No	No
Transmission Simplex	N/A	N/A
Transmission Half Duplex	N/A	N/A
Transmission Full Duplex	N/A	N/A

Features

Table 14-1. PCG Specifications (Cont'd)

Feature	PCGA-B	PCGC-D
Access Type		
Data Buffer	No	No
Core Data Access	N/A	N/A
DMA Data Access	N/A	N/A
DMA Channels	N/A	N/A
DMA Chaining	N/A	N/A
Boot Capable	N/A	N/A
Local Memory	No	No
Clock Operation	f_{PCLK}	f_{PCLK}

Features

The following list describes the features of the precision clock generators.

- Operates on the DAI and DPI units
- PCG input clock selection from CLKIN, PCLK or external DAI pins
- Provides 4 different clock dividers for serial clock, frame sync, phase (20-bit) and pulse width (16-bit)
- Phase shift allows adjustment of the frame sync relative to the serial clock and can be shifted the full period and wrap around
- Provides pulse width control for arbitrary frame sync signal generation

- Bypass mode for external frame sync manipulation
- External trigger mode starts PCG operation. No additional jitter introduced since operation is independent of the on-chip PLL by using off-chip clocks.

Pin Descriptions

[Table 14-2](#) provides the pin descriptions for the PCGs. Note x = unit A/B/C/D.

Table 14-2. PCG Pin Descriptions

Internal Nodes	I/O	Description
Inputs		
CLKIN	I	External clock input for PCG x
PCLK	I	Internal peripheral clock input for PCG x
PCG_SYNC_CLKx_I	I	External trigger used to enable the frame sync output
PCG_EXTx_I	I	External clock A input provided to the PCG x (not CLKIN)
MISCA2_I	I	External frame sync used for bypass mode PCG A
MISCA3_I	I	External frame sync used for bypass mode PCG B
MISCA4_I	I	External frame sync used for bypass mode PCG C
MISCA5_I	I	External frame sync used for bypass mode PCG D
Outputs		
PCG_CLKx_O	O	Serial clock x output
PCG_FSx_O	O	Frame sync x output

SRU Programming

To use the PCG, route the required inputs using the SRU as described [Table 14-3](#). Also, use the SRU to connect the outputs to the desired DAI pin.

Table 14-3. PCG DAI/SRU Connections

Internal Nodes	DAI Group	DPI Group	SRU Register
Inputs			
PCG_SYNC_CLKA_I PCG_SYNC_CLKB_I PCG_SYNC_CLKC_I PCG_SYNC_CLKD_I PCG_EXTA_I PCG_EXTB_I PCG_EXTC_I PCG_EXTD_I	Group A		SRU_CLK4 SRU_CLK5
MISCA2_I MISCA3_I MISCA4_I MISCA5_I	Group E		SRU_EXT_MISCA
Outputs			
PCG_CLKA_O PCG_CLKB_O	Group A, D Group A, D, E		
PCG_CLKC_O PCG_CLKD_O	Group D	Group B	
PCG_FSA_O PCG_FSB_O	Group C, D, E		
PCG_FSC_O PCG_FSD_O	Group D	Group B	



A PCG clock output cannot be fed to its own input. Setting SRU_CLK4[4:0] = 28 connects PCG_EXTA_I to logic low, not to PCG_CLKA_O. Setting SRU_CLK4[9:5] = 29 connects PCG_EXTB_I to logic low, not to PCG_CLKB_O. The clock and frame sync signals of

PCG C and D cannot be directly connected to other peripheral clock and frame sync signals. They can only be routed through the DAI pins.

Register Overview

The processor contains registers that are used to control the PCGs.

- **Control Register 0 (PCG_CTLx0).** Enables the clock and frame sync, it includes the frame sync divider and the upper half of the 20-bit phase value.
- **Control Register 1 (PCG_CTLx1).** Enables the clock and frame sources, it includes the clock divider and the lower half of the 20-bit phase value.
- **Pulse Width Register (PCG_PWx).** Contains the pulse width settings for normal mode ($\text{FSDIV} > 1$) or control bits for bypass mode ($\text{FSDIV} = 1/0$). Enables direct bypass or one shot mode.
- **Synchronization Register (PCG_SYNCx).** This register enables PCLK as input clock to the PCGs. It also enables external FS trigger mode.

Clocking

The fundamental clock of the PCG is PCLK .

Functional Description

The following sections provide information on the function of the precision clock generators.

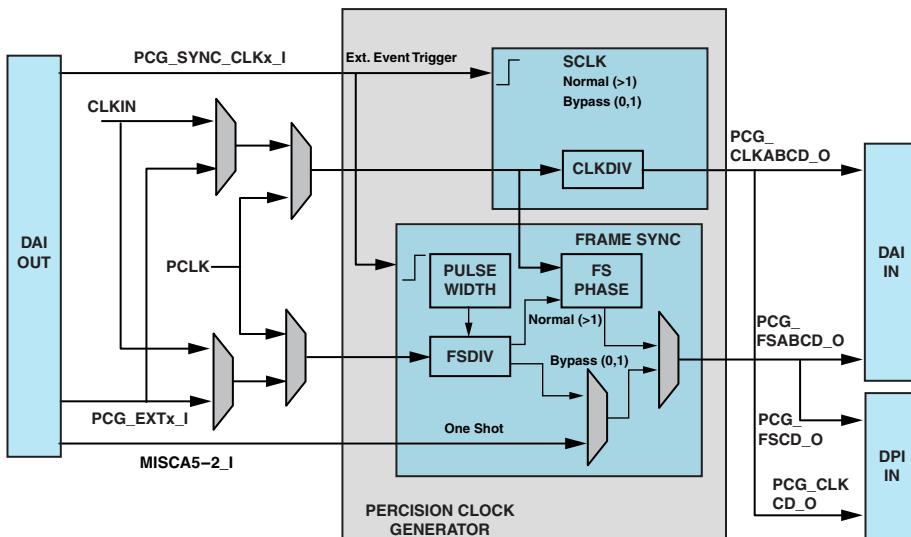


Figure 14-1. PCG Block Diagram

Serial Clock

Each of the four units (A, B, C, and D) produces a clock output. Serial clock generation from a unit is independently enabled and controlled. Sources for the serial clock generation can be either from the CLKIN, PCLK, or a DAI pin source. The clock output is derived from the input to the PCG with a 20-bit divisor.

Note that the divider is working in normal mode for $\text{CLKDIV} > 1$. For $\text{CLKDIV} = 0, 1$ the divider operates in bypass mode, (input clock is fed directly to its output). Note that in bypass mode, the clock at the output

can theoretically run at up to the `PCLK` frequency. However the DAI/DPI pin buffers limit the speed to `PCLK/4`.

Note that the clock output is always set (as closely as possible) to a 50% duty cycle. If the clock divisor is even, the duty cycle of the clock output is exactly 50%. If the clock divisor is odd, then the duty cycle is slightly less than 50%. The low period of the output clock is one input clock period more than the high period of the output clock. For higher values of an odd divisor, the duty cycle is close to 50%.



A PCG clock output cannot be fed to its own input.

Frame Sync

The following sections describe the use of frame syncs in the PCGs.

Frame Sync Output

Each of the four units (A through D) also produces a synchronization signal for framing serial data. The frame sync outputs are much more flexible since they need to accommodate the wide variety of serial protocols used by peripherals.

Frame sync generation from a unit is independently enabled and controlled. Sources for the frame sync generation can be either from the crystal buffer output, `PCLK`, or an external pin source. There is only one external source pin for both frame sync and clock output for a unit.

If an external source is selected for both frame sync and clock output for a unit, then they operate on the same input signal. Apart from enable and source select control bits, frame sync generation is controlled by a 20-bit divisor.

Functional Description

Divider Mode Selection

If frame sync divisor > 1 the PCG frame sync output frequency is equal to the input clock frequency, divided by a 20-bit integer. This integer is specified in the `FSDIV` bit field (bits 19–0 of the `PCG_CTLx0` register).

However if the frame sync divisor is zero or one, the PCG's frame sync clock generation unit is bypassed, and the frame sync input is connected directly to the frame sync output. For `FSDIV=0, 1` the `PCG_PWX` registers have different functionality than in normal mode.

Phase Shift

Phase shift is a frame sync parameter that defines the phase shift of the frame sync with respect to the input clock of the same unit. This feature allows shifting of the frame sync signal in time relative to the clock input signal. Frame sync phase shifting is often required by peripherals that need a frame sync signal to lead or lag a clock signal.

For example, the I²S protocol specifies that the frame sync transition from high to low occur one clock cycle before the beginning of a frame. Since an I²S frame is 64 clock cycles long, delaying the frame sync by 63 cycles produces the required framing.

Phase shifting is represented as a full 20-bit value so that even when the frame sync is divided by the maximum amount, the phase can be shifted to the full range, from zero to one input clock short of the period.



Phase shifting is specified as a 2 x 10-bit divider value in the `FSxPHASE_HI` bit field (bits 29–20) of the `PCG_CTLx0` register and in the `FSxPHASE_L0` bit field (bits 29–20) of the `PCG_CTLx1` register.

A single 20-bit value spans these two bit fields. The upper half of the word (bits 19–10) is in the `PCG_CTLx0` register, and the lower half (bits 9–0) is in the `PCG_CTLx1` register.

The phase shift between clock and frame sync outputs may be programmed using the `PCG_PW` and `PCG_CTLxx` registers under these conditions:

- The input clock source for the clock generator output and the frame sync generator output is the same.
- The clock and frame sync are enabled at the same time using a single atomic instruction.
- The frame sync divisor is an integral multiple of the clock divisor.



When using a clock and frame sync as a synchronous pair, the units must be enabled in a single atomic instruction before their parameters are modified. Both units must also be disabled in a single atomic instruction as shown below.

```
r0 = CLKDIV_A|PHASE_LO_A;
dm(PCG_CTLA1) = r0;
r0 = FSDIV_A|PHASE_HI_A|ENCLKA|ENFSA;
           /* program dividers and enable CLK and FS */
dm(PCG_CTLA0) = r0;
```

If the phase shift is zero (see [Figure 14-2](#)), the clock and frame sync outputs rise at the same time. If the phase shift is one, the frame sync output transitions one input clock period ahead of the clock transition. If the phase shift is divisor – 1, the frame sync transitions divisor – 1 input clock periods ahead of the clock transitions

Pulse Width

Pulse width is the number of input clock periods for which the frame sync output is high.

A 16-bit value determines the width of the framing pulse. Settings for pulse width can range from zero to `DIV` – 1. The pulse width should be less

Functional Description

than the divisor of the frame sync. The pulse width of frame sync is specified in the `PWFSx` bits (15–0) and (31–16) of the `PCG_PWx` registers.

Default Pulse Width

If the pulse width count is equal to 0 and if `FSDIV` bit field is even, then the actual pulse width of the frame sync output is equal to:

For even divisors: frame sync divisor/2

If the pulse width count is equal to 0 and if `FSDIV` bit field is odd, then the actual pulse width of the frame sync output is equal to:

For odd divisors: frame sync divisor – 1/2

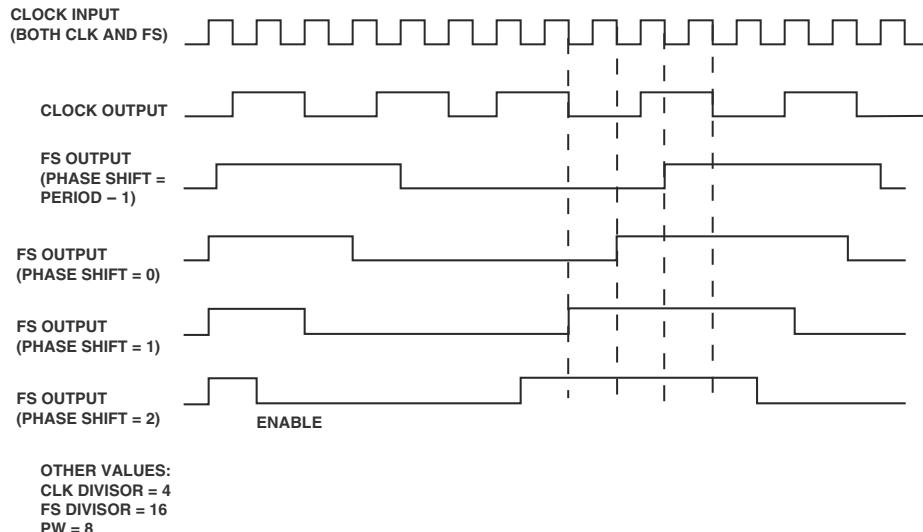


Figure 14-2. Phase and Pulse Width Settings

Timing Example for I²S Mode

For I²S mode, the frame sync should be driven at the falling edge of SCLK. In other words, the frame sync edge should coincide with the falling edge of the SCLK. To satisfy this requirement, the phase of the frame sync should be programmed accordingly in the PCG_CTLxx registers.

For example, assume that the input clock source for both clock and frame sync are the same and both the clock and frame sync are enabled at the same time. Also assume that the clock divisor value needed to generate the required SCLK is CLKDIV = 4. Then, for a 32-bit word length, the frame sync divisor value should be FSDIV = $64 \times \text{CLKDIV} = 256$.

By default, for phase = 0, the rising edge of both SCLK and frame sync will coincide. To make sure that the frame sync edges coincides with the falling edge of the SCLK, the phase value needs to be programmed as CLKDIV/2 = 2. It can be done by following instructions:

```
ustat1=CLKDIV|((CLKDIV/2) << 20);  
dm(PCG_CTLx1) = ustat1;
```

For details on how to program phase of the frame sync see “[Programming Model](#)” on page 14-20.

Operating Modes

The following sections provide information on the operating modes of the precision clock generator.

Operating Modes

Normal Mode

When the frame sync divisor is set to any value other than zero or one, the PCGs operates in normal mode. In normal mode, the frequency of the frame sync output is determined by the divisor where:

$$\text{Frequency of Frame Sync Output} = \left(\frac{\text{Input Frequency}}{\text{Divisor}} \right)$$

The high period of the frame sync output is controlled by the value of the pulse width control. The value of the pulse width control should be less than the value of the divisor.

The phase of the frame sync output is determined by the value of the phase control. If the phase is zero, then the positive edges of the clock and frame sync coincide when:

- the clock and frame sync dividers are enabled at the same time using an atomic instruction
- the divisors of the clock and frame sync are the same
- the source for the clock and frame sync is the same

The number of input clock cycles that have already elapsed before the frame sync is enabled is equal to the difference between the divisor and the phase values. If the phase is a small fraction of the divisor, then the frame sync appears to lead the clock. If the phase is only slightly less than the frame sync divisor, then the frame sync appears to lag the clock. The frame sync phase should not be greater than the divisor.

Bypass Mode

When the frame sync divisor for the frame sync has a value of zero or one, the frame sync is in bypass mode, and the PCG_PWx registers have different functionality than in normal mode.



In normal mode bits 15–0 and 31–18 of the PCG_PWx registers are used to program the pulse width count. In bypass mode bits 15–2 and 31–18 are ignored. Bits 1–0 and 17–16 are renamed to STROBEx and INFSx respectively. This is described in more detail below.

If the STROBEx bit of PCG_PWx register is cleared, then the input is directly passed (see [Figure 14-3](#)) to the frame sync output either inverted or not inverted, depending on the INVFSx bit of the PCG_PWx registers.

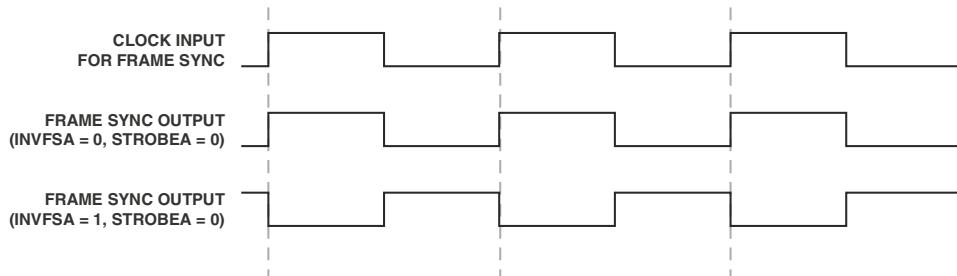


Figure 14-3. Bypass and Inverted Bypass

One-Shot Mode

In one-shot mode operation, the PCG produces a series of periods but does not run continuously.

Bypass mode also enables the generation of a strobe pulse (one shot frame sync). Strobe usage ignores the divider counters and looks to the SRU to provide the input signal. Two bit fields determine the operation in this mode.

Operating Modes

In the bypass mode, if the STROBEx bit of PCG_PWx register is set to 1, then a one-shot pulse is generated. This one-shot pulse has the duration equal to the period of MISCAx_I for the PCGx unit. This pulse is generated either at the falling or rising edge of the input clock, depending on the value of the INVFSx bit of the PCG_PW register. The output pulse width is equal to the period of the SRU source signal MISCAx_I. The pulse begins at the second rising edge of MISCAx_I following a rising edge of the clock input. When the INVFSx bit is set, the pulse begins at the second rising edge of MISCAx_I coinciding with or following a falling edge of the clock input.



Notice a strobe period is defined to be the period of the FS input clock signal specified by the FSxSOURCE bit (PCG_CTLx1 registers).

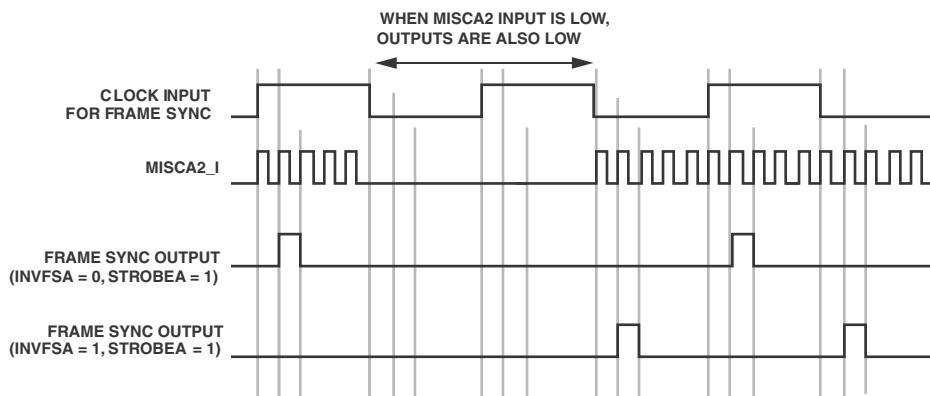


Figure 14-4. One Shot Mode PCG A (MISCA2_I input)

External Event Trigger

The trigger with the external clock is enabled by setting bits 0 and 16 of the PCG_SYNC register.

Since the rising edge of the external clock is used to synchronize with the frame sync, the frame sync output is not generated until a rising edge of the external clock is sensed (Figure 14-5).

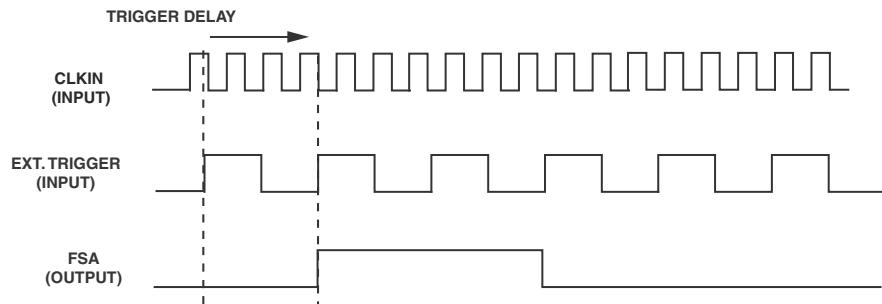


Figure 14-5. FS Output Synchronization With External Trigger Input

External Event Trigger Delay

The time delay between the rising trigger edge and the start of `SCLK/FS` varies between 2.5 to 3.5 input clock periods. If the input clock and the trigger signal are synchronous, the delay is 3 input clock periods. The following cases need to be considered:

- `PCLK` is the input source. In this case if the given trigger event is synchronous to `PCLK`, the delay is 3 `PCLK` periods. If the trigger signal is asynchronous with `PCLK`, the delay varies from 2.5 `PCLK` periods to 3.5 `PCLK` periods. (It depends on whether the trigger edge occurs in the positive half cycle or negative half cycle of `PCLK`.)
- `CLKIN` is the input source. In this case if the given trigger signal is synchronous to `CLKIN`, the delay is 3 `CLKIN` periods. But if they are asynchronous to `CLKIN`, the delay can vary between 2.5 `CLKIN` periods to 3.5 `CLKIN` periods.

Operating Modes

- SRU is the input source. If the input clock and trigger signal are synchronous, the delay is exactly 3 input clock periods. If asynchronous, it varies between 2.5 to 3.5 input clock periods depending on the phase difference between the input clock and trigger signal.

Audio System Example

Figure 14-6 shows an example of the internal interconnections between the SPDIF receiver, ASRC, and the PCGs. The interconnections are made by programming the signal routing unit.

It shows how to set up two precision clock generators using the S/PDIF receiver and an asynchronous sample rate converter (ASRC) to interface to an external audio DAC. The PCG is configured to provide a fixed ASRC/DAC output sample rate of 65.098 kHz. The input to the S/PDIF receiver is typically 44.1 kHz if supplied by a CD player, but can also be from other source at any nominal sample rate from about 22 kHz to 192 kHz.

Similarly, the phase shift for frame syncs B, C, and D is specified in the corresponding `PCG_CTLx0` and `PCG_CTLx1` registers.

Three synchronous clocks are required in audio systems

1. Frame sync (FS)
2. Serial bit clock (64 x FS)
3. Master DAC clock (256 x 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 `FS`, these two outputs should come from one PCG (PCG A) while the master clock comes from the 2nd (PCG B).

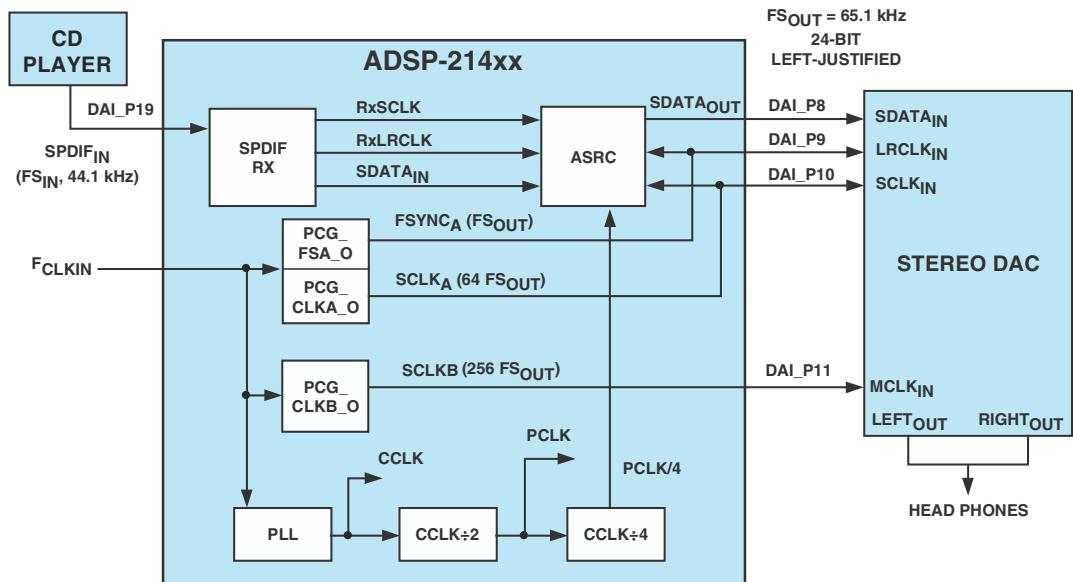


Figure 14-6. PCG Setup for I2S or Left-Justified DAI

The combined PCGs can provide a selection of synchronous clock frequencies to support alternate sample rates for the ASRCs and external DACs. However, the range of choices is limited by CLKIN and the ratio of PCG_CLKx_0:SCLK:FS 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 786x FS for PCG_CLKx_0, which allows some additional flexibility in choosing CLKIN.

Note the falling edge of SCLK must always be synchronous with both edges of FS. This requires that the phase of the SCLK and FS signals for a common PCG (PCG A) be adjustable.

While the frequency of the master DAC clock (PCG_CLKx_0) must be synchronous with the sample rate supplied to the external DAC, there is no fixed phase requirement.

Operating Modes

Set the clock divisor and source and low-phase word first, followed by the control register enable bits, which must be set together. When the PCG_PW register is set to zero (default) the FS pulse width is (divisor ÷ 2) for even divisors and (divisor – 1) ÷ 2 for odd divisors. Alternatively, the PCG_PW register could be set high for exactly one-half the period of CLKIN cycles for a 50% duty cycle, provided the FS divisor is an even number.

Clock Configuration Examples

For a CLKIN = 33.330 MHz the two PCGs provide the three synchronous clocks PCGx_CLK, SCLK and FS for the SRCs and external DAC. These divisors are stored in 20-bit fields in the PCG_CTL registers.

The integer divisors for several possible sample rates based on 33.330 MHz CLKIN are shown in [Table 14-4](#).

Table 14-4. Precision Clock Generator Division Ratios
(33.330 CLKIN)

Sample Rate kHz	PCG Divisors		
	CLKDIV B	CLKDIV A	FSDIV A ¹
130.195	1	4	256
65.098	2	8	512
43.398	3	12	768
32.549	4	16	1024
26.039	5	20	1280
21.699	6	24	1536
18.599	7	28	1792

¹ The frame sync divisor should be an even integer in order to produce a 50% duty cycle waveform. See [“Frame Sync” on page 14-7](#).

For more information on core clock setting, see “[ADSP-2146x Power Management Registers](#)” on page [A-6](#) and “[ADSP-2147x/ADSP-2148x Power Management Registers](#)” on page [A-12](#).

Effect Latency

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

Write Effect Latency

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

PCG Effect Latency

After the PCG registers are configured the effect latency is shown below. The latency to start the `CLKOUT` depends on the divisor value and input source as described below.

Input clock through `PCLK`

- If divisor value is 0 or 1 (bypassed) the latency is 1 `PCLK` cycle
- For other divisor values the latency is 3 `PCLK` cycles

Input clock through `CLKIN`

- If divisor is 0 or 1 (bypassed) the latency can vary from 0 to 1 oscillator period. This is because clock generation starts with the immediate positive edge of the `CLKIN`.
- For other divisor values the latency can vary between 2 to 3 oscillator periods. This is because clock generation starts with the third positive edge of `CLKIN`.

Programming Model

Input clock through SRU

- If divisor is 0 or 1 (bypassed) the latency can vary from 0 to 1 input clock period. For example if the input clock has a period of 100 ns then this latency can be a maximum of 100 ns.
- For other divisor values the latency can vary between 2 to 3 input clock periods. For example if the input clock has a period of 100 ns then this latency can be between 200 and 300 ns.

Programming Model

The section describes which sequences of software steps required for successful PCG operation.

Frame Sync Phase Setting

The phase unit requires that the clock and FS is enabled simultaneously in an atomic instruction.

1. Write the clock divider/low 10-bit Phase divider to `PCG_CTLx1` register.
2. Program the FS divider/high 10-bit Phase divider, enable both the `ENCLKx` and `ENFSx` bits in the `PCG_CTLx0` registers.

Note that both units must be disabled in the same way.

External Event Trigger

The trigger with the external clock is enabled by setting bits 0 and 16 of the `PCG_SYNC` register. The phase must be programmed to 3, so that the rising edge of the external clock is in sync with the frame sync ([Figure 14-5](#)).

Programming should occur in the following order.

1. Program the PCG_SYNC and the PCG_CTLA0-1, PCG_CTLB0-1 registers appropriately.
2. Enable clock or frame sync, or both.

Since the rising edge of the external clock is used to synchronize with the frame sync, the frame sync output is not generated until a rising edge of the external clock is sensed.

Debug Features

Care should be taken in cases where any input to the phase unit is modified. Any individual change of the CLKDIV or FSDIV dividers may cause a failure in PCG sync operation between the serial clock and the frame sync. Only the programming model ensures a correct setup for phase settings.

Debug Features

15 SERIAL PERIPHERAL INTERFACE PORTS

The ADSP-214xx processors are equipped with two synchronous serial peripheral interface ports that are compatible with the industry-standard serial peripheral interface (SPI). Each SPI port also has its own set of registers (the secondary register set contains a B as in `SPIBAUDB`). The SPI ports support communication with a variety of peripheral devices including codecs, data converters, sample rate converters, S/PDIF or AES/EBU digital audio transmitters and receivers, LCDs, shift registers, microcontrollers, and FPGA devices with SPI emulation capabilities. The interface specifications are shown in [Table 15-1](#).

Table 15-1. SPI Port Specifications

Feature	SPI/SPIB
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	Yes
SRU2 DPI Default Routing	Yes
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	Yes
Transmission Half Duplex	Yes

Features

Table 15-1. SPI Port Specifications (Cont'd)

Feature	SPI/SPIB
Transmission Full Duplex	Yes (Core and DMA)
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	1
DMA Chaining	Yes
Interrupt Source	Core/DMA
Boot Capable	Yes
Local Memory	No
Clock Operation	$f_{PCLK}/4$ (slave) $f_{PCLK}/8$ (master)

Features

The processor's SPI ports provide the following features and capabilities.

- A simple 4-wire interface consisting of two data pins, a device select pin, and a clock pin.
- Special data formats to accommodate little and big endian data, different word lengths, and packing modes.
- Master and multiples slave (multi devices) in which the ADSP-214xx master processor can be connected to up to four other SPI devices.
- Parallel core and DMA access allow full duplex operation.
- Open drain outputs to avoid data contention and to support multimaster scenarios.

- Programmable baud rates, clock polarities, and phases (SPI mode 0–3).
- Master or slave booting from a master SPI device. See “[SPI Port Booting](#)” on page 23-12.
- DMA capability to allow transfer of data without core overhead. See “[DMA Transfers](#)” on page 15-21.
- Internal loopback mode (by connecting MISO to MOSI).

Note the SPI interface does not support daisy chain operation, where the MOSI and MISO pins are internally connected through a FIFO, allowing bypass of data streams.

Pin Descriptions

The SPI protocol uses a 4-wire protocol to enable full-duplex serial communication. [Table 15-2](#) provides detailed pin descriptions and [Figure 15-1](#) shows the master-slave connections between two devices.

Table 15-2. SPI Pin Descriptions

Internal Node	Type	Description
SPI_CLK_I/O SPIB_CLK_I/O	I/O	SPI Clock Signal. This control line is clock driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of baud rates. The CLK line cycles once for each bit that is transmitted. It is an output signal if the device is configured as a master; it is an input signal if configured as a slave.
SPI_DS_I SPIB_DS_I	I	SPI Slave Device Select. This is an active-low input signal that is used to enable slave devices. This signal is like a chip select signal for the slave devices and is provided by the master device. For a master device, it can act as an error input signal in a multi-master environment. In multi-master mode, if the /SPIDS input signal of a master is asserted (Low) an error has occurred. This means that another device is also trying to be the master.

SRU Programming

Table 15-2. SPI Pin Descriptions (Cont'd)

Internal Node	Type	Description
SPI_MOSI_I/O SPIB_MOSI_I/O	I/O	SPI Master Out Slave In. This data line transmits the output data from the master device and receives the input data to a slave device. This data is shifted out from the MOSI pin of the master and shifted into the MOSI input(s) of the slave(s).
SPI_MISO_I/O SPIB_MISO_I/O	I/O	SPI Master In Slave Out. This data line transmits the output data from the slave device and receives the input data to the master device. This data is shifted out from the MISO pin of the slave and shifted into the MISO input of the master. There may be no more than one slave that is transmitting data during any particular transfer.
SPI_FLG3-0_O SPIB_FLG3-0_O	O	SPI Slave Select Out. The slave select pins are used to address up to 4 slaves in a multi device system. This functionality can be routed to any of the DPI pins. This frees up the multiplexed core flags for other purposes.
SPI_CLK_PBEN_O SPIB_CLK_PBEN_O SPI_MOSI_PBEN_O SPIB_MOSI_PBEN_O SPI_MISO_PBEN_O SPIB_MISO_PBEN_O SPI_FLG3-0_PBEN_O SPIB_FLG3-0_PBEN_O	O	SPI Pin buffer Enable Out Signal. Only driven in master mode. The SPIx_FLGx_PBEN_O signals are enabled if the corresponding DSxEN bits in the SPIFLAG register are set.

SRU Programming

Both SPI and SPIB signals are available through the SRU2, and are routed as described in [Table 15-3](#).

Since the SPI supports a gated clock, it is recommended that programs enable the SPI clock output signal with its related pin buffer enable. This can be done using the macro SRU (SPI_CLK_PBEN_0, PBEN_03_I). If these signals are routed statically high as in SRU (high, PBEN_03_I) some SPI timing modes that are based on polarity and phase may not work correctly because the timing is violated.

Table 15-3. SPI DPI/SRU2 Signal Connections

Internal Node	DPI Group	SRU2 Register
Inputs		
SPI_CLK_I SPIB_CLK_I SPI_DS_I SPIB_DS_I SPI_MOSI_I SPIB_MOSI_I SPI_MISO_I SPIB_MISO_I	Group A	SRU2_INPUT1-0
Outputs		
SPI_CLK_O SPIB_CLK_O SPI_MOSI_O SPIB_MOSI_O SPI_MISO_O SPIB_MISO_O SPI_FLG3-0_O SPIB_FLG3-0_O	Group B	
SPI_CLK_PBEN_O SPIB_CLK_PBEN_O SPI_MOSI_PBEN_O SPIB_MOSI_PBEN_O SPI_MISO_PBEN_O SPIB_MISO_PBEN_O SPI_FLG3-0_PBEN_O SPIB_FLG3-0_PBEN_O	Group C	

Register Overview

This section provides brief descriptions of the major registers. For complete information see “Serial Peripheral Interface Registers” on page [A-232](#).

Clocking

SPI Control (SPICTLx). This register configures the fundamental transfer initiation mode (core or DMA) and configure timing bits and enable the SPI port.

SPI DMA Control (SPIDMACx). This register control the DMA channel on SPI and corresponding status bits provide status or error information on transmission.

SPI Flag (SPIFLAGx). This control register enables the chip selects output in master mode and returns status for errors in multiprocessor systems.

SPI Status (SPISTATx). This status register provide information on transmission errors for the core.

SPI Baud rate (SPIBAUDx). For master devices, the clock rate is determined by the 15-bit value of the baud rate registers (SPIBAUDx) as shown in [Table 15-4](#). For slave devices, the value in the SPIBAUDx register is ignored.

Clocking

The fundamental timing clock of the SPI module is peripheral clock/4 ($\text{PCLK}/4$) for slave mode and peripheral clock/8 ($\text{PCLK}/8$) for master mode. In master mode the settings define the SPI master clock. The baud rate, ssettings are shown in [Table 15-4](#).

Table 15-4. SPI BAUD Rate – PCLK = 200 MHz

BAUDR Bit Setting	Divider	SPICLK
0	N/A	N/A
1	N/A	N/A
2	8	25
3	12	16.66

Table 15-4. SPI BAUD Rate – PCLK = 200 MHz (Cont'd)

BAUDR Bit Setting	Divider	SPICLK
4	16	12.5
25	100	2.0 (master boot)
32,767	131068	1526 Hz

Choosing the Pin Enable for the SPI Clock

When using the SPI in master mode, and the `SPIxCLK` signal is routed onto the DPI pin, then the `DPI_PBENxx_I` signal for that DPI pin being used for the clock must be connected to high.

However, depending on the SPI mode being used (based on the setting of `CPHASE` and `CLKPL` bits in the `SPICTL` register), `SPIx_CLK_PBEN_O` signal may be used.

Choosing the correct pin enable ensures that the very first edge on `SPIx_CLK` (DPI pin) output is not incorrectly chosen as a sampling edge by the slave SPI. [Table 15-5](#) shows the correct pin enable to use for a chosen SPI mode.

Table 15-5. Pin Enable Selection by Mode

Mode	CLKPL	CPHASE	Pin Enable
0	0	0	HIGH
1	0	1	HIGH
2	1	0	<code>SPIx_CLK_PBEN_O</code>
3	1	1	<code>SPIx_CLK_PBEN_O</code>

All other SPI signals `SPIx_MOSI`, `SPIx_MISO` and `SPIx_FLGx` signals when routed on the DPI pins, the `SPIx_MISO_PBEN_O`, `SPIx_MOSI_PBEN_O`, or `SPIx_FLG_PBEN_O` signals should be connected to corresponding `DPI_PBENxx_I` signals. The `DPI_PBENxx_I` signals should not be statically

Functional Description

connected to high, as it affects the functioning of certain bits in the `SPICTLx` register.

Functional Description

Each SPI interface contains its own transmit shift (`TXSR`, `TXSRB`) and receive shift (`RXSR`, `RXSRB`) registers (not user accessible). The `TXSRx` registers serially transmit data and the `RXSRx` registers receive data synchronously with the SPI clock signal (`SPICLK`). [Figure 15-1](#) shows a block diagram of the SHARC processor SPI interface. The data is shifted into or out of the shift registers on two separate pins: the master in slave out (`MISO`) pin and the master out slave in (`MOSI`) pin.

During data transfers, one SPI device acts as the SPI master by controlling the data flow. It does this by generating the `SPICLK` and asserting the SPI device select signal (`SPIDS`). The SPI master receives data using the `MISO` pin and transmits using the `MOSI` pin. The other SPI device acts as the SPI slave by receiving new data from the master into its receive shift register using the `MOSI` pin. It transmits requested data out of the transmit shift register using the `MISO` pin.

Each SPI port contains a dedicated transmit data buffer (`TXSPI`, `TXSPIB`) and a receive data buffer (`RXSPI`, `RXSPIB`). Transmitted data is written to `TXSPIx` and then automatically transferred into the transmit shift register. Once a full data word has been received in the receive shift register, the data is automatically transferred into `RXSPIx`, from which the data can be read. When the processor is in SPI master mode, programmable flag pins provide slave selection. These pins are connected to the `SPIDS` of the slave devices.

The SPI has a single DMA engine which can be configured to support either an SPI transmit channel or a receive channel, but not both simultaneously. Therefore, when configured as a transmit channel, the received data is essentially ignored. When configured as a receive channel, what is

transmitted is irrelevant. A 4-word deep FIFO is included to improve throughput on the IOD0 bus.

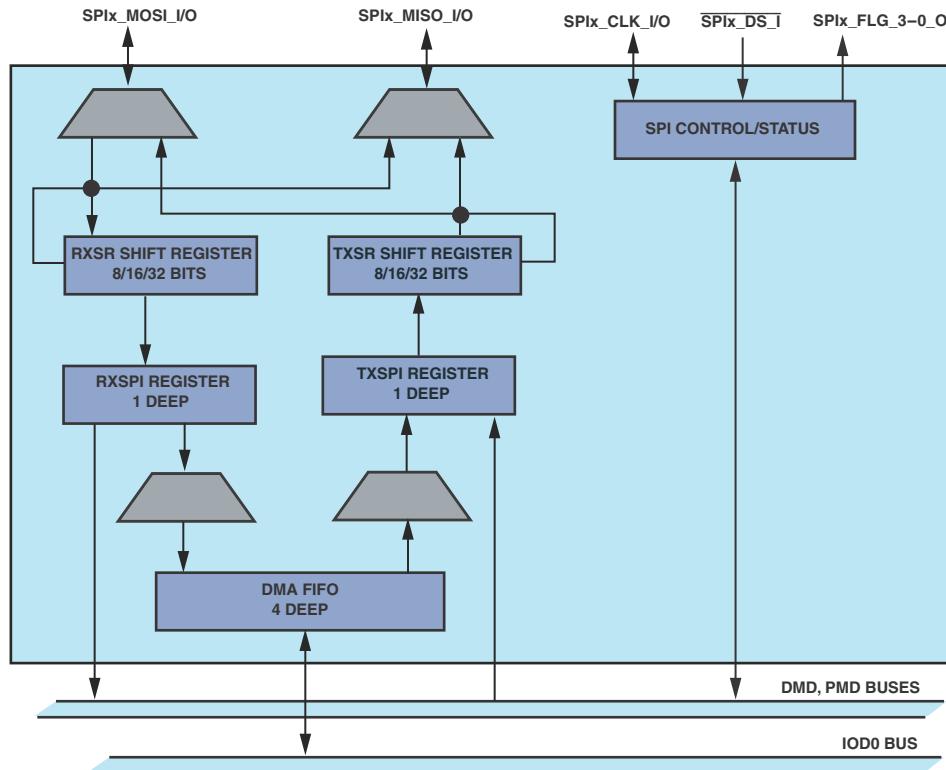


Figure 15-1. SPI Block Diagram

SPI Transaction

An SPI transaction defined start and end depend on whether the device is configured as a master or a slave, whether CPHASE mode is selected, and whether the transfer initiation mode is (**TIMOD**) selected. For a master SPI with **CPHASE** = 0, a transfer starts when either the **TXSPI** register is written or the **RXSPI** register is read, depending on the **TIMOD** selection. At the start of the transfer, the enabled slave-select outputs are driven active (low).

Functional Description

However, the SPICLK starts toggling after a delay equal to one-half (0.5) the SPICLK period. For a slave with CPHASE = 0, the transfer starts as soon as the SPI_DS_I input transitions to low.

For CPHASE = 1, a transfer starts with the first active edge of SPICLK for both slave and master devices. For a master device, a transfer is considered complete after it sends and simultaneously receives the last data bit. A transfer for a slave device is complete after the last sampling edge of SPICLK.

Single Master Systems

[Figure 15-2](#) illustrates how the SHARC processor can be used as the slave SPI device. The 16-bit host (A Blackfin ADSP-BF53x processor) is the SPI master. The processor can be booted via its SPI interface to allow application code and data to be downloaded prior to runtime.

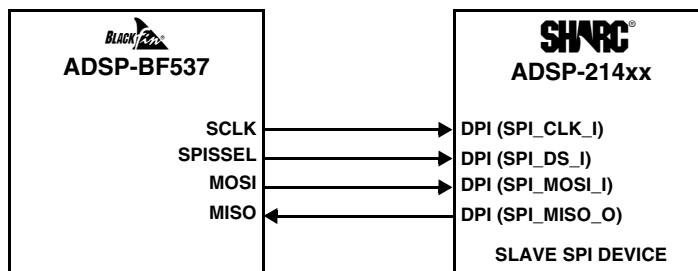


Figure 15-2. SHARC Processor as SPI Slave

[Figure 15-3](#) shows an example SPI interface where the SHARC processor is the SPI master. With the SPI interface, the processor can be directed to alter the conversion resources, mute the sound, modify the volume, and power down the AD1855 stereo DAC.

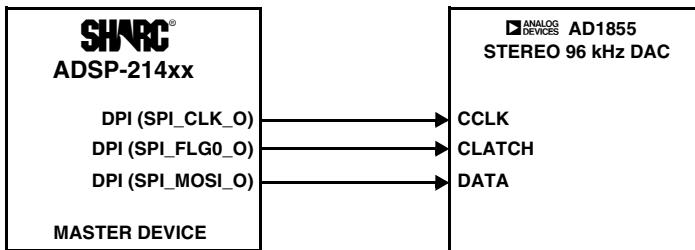


Figure 15-3. SHARC Processor as SPI Master

Multi Master Systems

The SPI does not have an acknowledgement mechanism to confirm the receipt of data. Without a communication protocol, the SPI master has no knowledge of whether a slave even exists. Furthermore, the SPI has no flow control.

Slaves can be thought of as input/output devices of the master. The SPI does not specify a particular higher-level protocol for bus mastership. In some applications, a higher-level protocol, such as a command-response protocol, may be necessary. Note that the master must initiate the frames for both its' command and the slave's response.

Multi master mode allows an SPI system to transfer mastership from one SPI device to another. In a multi device SPI configuration, several SPI ports are connected and any one (but only one) of them can become a master at any given time.

In this configuration, every MOSI pin in the SPI system is connected. Likewise, every MISO pin in the system is on a single node, and every SPI CLK pin should be connected (see [Figure 15-4](#)). SPI transmission and reception are always enabled simultaneously, unless the broadcast mode has been selected.

The master's FLAG_x pins connect to each of the slave SPI devices in the system via their SPI DS pins. To enable the different slaves, connect the slave

Operating Modes

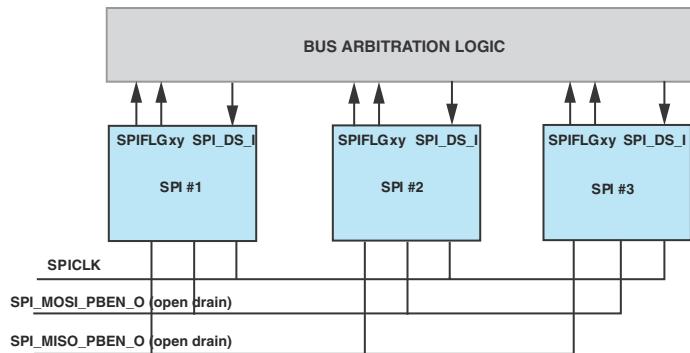


Figure 15-4. Multi Master System

SPIDS pins to the DPI pins of the master SHARC. Since these flags are NOT open drain, slave select pins cannot be shorted together in multi master environment. To control slave selects, an external glue logic is required in a multi-master environment.

Another feature is implemented to troubleshoot the bus mastership protocol. If a recent SHARC bus master receives an invalidly asserted SPIDS signal, it triggers an error handling scenario using the MME bit (SPIMME bit for DMA) and ISSEN bit to reconfigure the SPI to slave mode, and jump into an ISR. This ensures that any potential driver conflict is solved. [For more information, see “Control Registers \(SPICTL, SPICTLB\)” on page A-232.](#)

Operating Modes

This section describes the different mechanisms used for master or slave select operation modes.

Transfer Initiate Mode

When the processor is enabled as a master, the initiation of a transfer is defined by the `TIMOD` bits (1–0). Based on these two bits and the status of the interface, a new transfer is started upon either a read of the `RXSPIX` registers or a write to the `TXSPIX` registers. This is summarized in [Table 15-6](#).

Table 15-6. Transfer Initiation

<code>TIMOD</code>	Function	Transfer Initiated Upon	Action, Interrupt
00	Core Receive and Transmit	Initiate new single word transfer upon read of RXSPI and previous transfer completed. The SPICLK is generated after the data is read from the buffer. In this configuration, a dummy read is needed initially to receive all the data transmitted from the transmitter.	The SPI interrupt is latched in every core clock cycle in which the RXSPI buffer has a word in it. Emptying the RXSPI buffer or disabling the SPI port at the same time (<code>SPIEN</code> = 0) stops the interrupt latch.
01	Core Transmit and Receive	Initiate new single word transfer upon write to TXSPI and previous transfer completed.	The SPI interrupt is latched in every core clock cycle in which the TXSPI buffer is empty. Writing to the TXSPI buffer or disabling the SPI port at the same time (<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 TXSPI or a DMA read of RXSPI depending on the direction of the transfer as specified by the SPIRCV bit.	If chaining is disabled, the SPI interrupt is latched in the cycle when the DMA count decrements from 1 to 0. If chaining is enabled, interrupt function is based on the PCI bit in the CP register. If <code>PCI</code> = 0, the SPI interrupt is latched at the end of the DMA sequence. If <code>PCI</code> = 1, then the SPI interrupt is latched after each DMA in the sequence.
11	Reserved		

SPI Modes

The SPI supports four different combinations of serial clock phases and polarity called SPI modes. The application code can select any of these combinations using the CLKPL and CPHASE bits (10 and 11).

[Figure 15-5 on page 15-15](#) shows the transfer format when CPHASE = 0 and [Figure 15-6 on page 15-16](#) 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 SPIDS signal represents the slave device select input to the processor from the SPI master. The diagrams represent 8-bit transfers ($WL = 0$) with MSB first ($MSBF = 1$). Any combination of the WL and MSBF bits of the SPICTL register is allowed. For example, a 16-bit transfer with the LSB first is one possible configuration.

The clock polarity and the clock phase should be identical for the master device and slave devices involved in the communication link. The transfer format from the master may be changed between transfers to adjust to various requirements of a slave device.



When CPHASE = 0, the slave-select line, SPIDS, must be inactive (HIGH) between each word in the transfer. Even in SPI slave mode when CPHASE = 0, the master should de assert the SPIDS line between each transfer. When CPHASE = 1, SPIDS may either remain active (LOW) between successive transfers or be inactive (HIGH).

[Figure 15-5](#) shows the SPI transfer protocol for CPHASE = 0. Note that SPICLK starts toggling in the middle of the data transfer where the bit settings are $WL = 0$, and $MSBF = 1$.

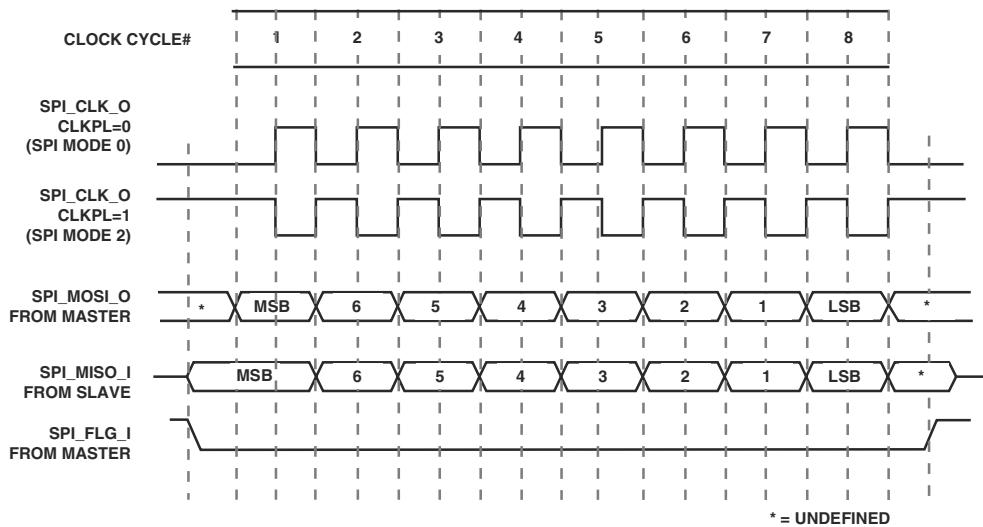


Figure 15-5. SPI Transfer Protocol for CPHASE = 0

[Figure 15-6](#) shows the SPI transfer protocol for CPHASE = 1. Note that SPICLK starts toggling at the beginning of the data transfer where the bit settings are WL = 0, and MSBF = 1.

Slave Select Outputs

If the SPI is enabled and configured as a master, any of the 14 DPI I/O pins may be used as slave-select outputs. For each DS_xEN bit which is set in the SPIFLG register, the corresponding SPI_FLG_x_O is configured as a slave-select output.

For example, if DS1EN = 1 is set, SPI_FLG1_O is driven as a slave-select. At the chip-level, SPI_FLG1_O can be connected to any of the DPI pins through SRU programming. For those DS_xEN bits which are not set, the corresponding SPI_x_FLG_x_PBEN_O is driven low.

The behavior of the SPI_FLG_x output depends on the value of the CPHASE configuration bit. If CPHASE = 1, all selected outputs may either remain

Operating Modes

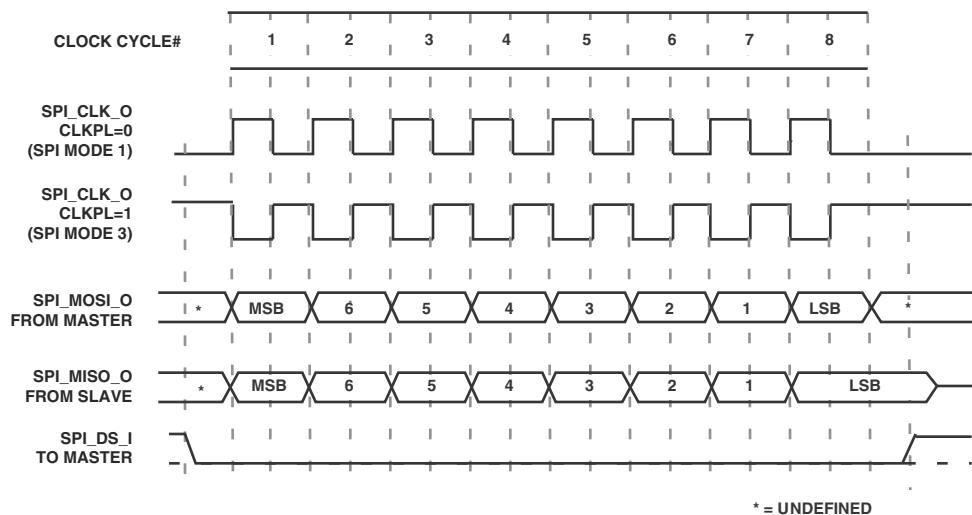


Figure 15-6. SPI Transfer Protocol for CPHASE = 1

asserted (active-low) between transfers or be deasserted between transfers. This is controlled in software using the SPIFLG_x bits (SPIFLG register). For example, to configure SPI_FLG1_0 as a slave-select, set DS1EN = 1 and SPIFLG1 = 1. As soon as this SPIFLG register write takes effect, the SPI_FLG1_0 (slave-select output pin) becomes active (Low).

If needed, SPI_FLG_x_0 can be cycled high and low between transfers by setting the SPIFLG[x] bit to 1 and back to 0. Otherwise, SPI_FLG_x_0 remains active between transfers.

If CPHASE = 0 or CPHASE = 1 and AUTOSDS = 1, all selected outputs are asserted only for the duration of the transfer. This is controlled by the internal SPI hardware. In this case, the SPIFLG_x bits are ignored. For example, to configure SPI_FLG1_0 as a slave-select, it is only necessary to set DS1EN=1.

Note that the SPI_FLG_x_0 signals behave as slave-select outputs only if the SPI module is enabled as a master. Otherwise, none of the bits in the SPIFLG register have any effect.

Variable Frame Delay for Slave

When the processor is configured as an SPI slave, the SPI master must drive an `SPICLK` signal that conforms with [Figure 15-7](#). For exact timing parameters, please refer to the appropriate product data sheet.

As shown in [Figure 15-7](#), the `SPIDS` lead time (T1), the `SPIDS` lag time (T2), and the sequential transfer delay time (T3) must always be greater than or equal to one-half the `SPICLK` period. The minimum time between successive word transfers (T4) is two `SPICLK` periods. This time period is measured from the last active edge of `SPICLK` of one word to the first active edge of `SPICLK` of the next word. This calculation is independent from the configuration of the SPI (`CPHASE`, `SPIMS`, and so on).

This is shown as:

$$T4 = 1.5 \text{ SPI clock period} + T3$$

and

$$T3 = 0.5 \text{ SPICLK period for } STDC = 0.$$

$$T3 = STDC \times \text{SPICLK period for } STDC > 0.$$



Unlike previous SHARC processors, a variable frame delay is included to increase SPI timing flexibility.

For a master device with `CPHASE` = 0 or `CPHASE` = 1 (with `AUTODS` set to 1 in the `SPCTL` register), this means that the slave-select output is inactive (high) for at least one-half the `SPICLK` period. In this case, T1 and T2 are each always be equal to one-half the `SPICLK` period.

Data Transfers

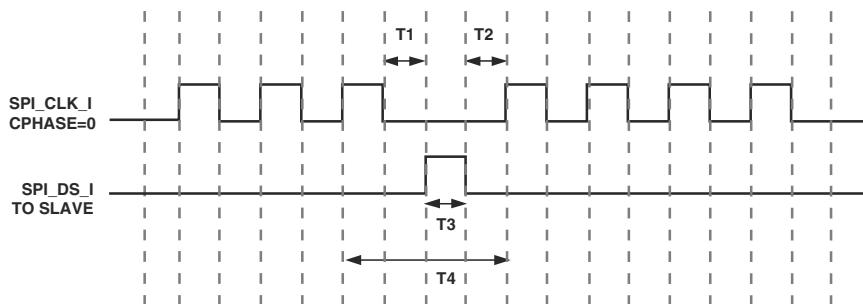


Figure 15-7. SPICLK Timing

When word to word delay is enabled ($WTWDEN = 1$) in the `SPICTL` register, then T_3 may vary with respect to the value programmed using the `STDC` bits in the `SPIBAUD` register. So the word to word delay T_4 is:

This is shown as:

$$T_4 = 1.5 \text{ SPI clock period} + T_3 \text{ and}$$

$$T_3 = 1.5 \text{ SPI clock period for } STDC = 0, BAUDR = 1, \text{ RX master}$$

$$T_3 = 0.5 \text{ SPI clock period for } STDC = 0, \text{ in all other cases.}$$

$$T_3 = STDC \times \text{SPI clock period for } STDC > 0.$$

Data Transfers

The SPI is capable of transferring data via the core and DMA. The following sections describe these transfer types.

Buffers

The SPI allows 3 different word lengths, which impacts the transmit or receive buffers with different packing methods.

8-bit word. The SPI port sends out only the lower eight bits of the word written to the SPI buffer. For example, 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. This code works only if the MSBF bit is zero in both the transmitter and receiver, and the SPICLK frequency is less. If MSBF = 1 in the transmitter and receiver, and SPICLK has a lower frequency, the received words follow the order 0x12, 0x34, 0x56, 0x78.

16-bit word. When transmitting, the SPI port sends out only the lower 16 bits of the word written to the SPI buffer. When receiving, the SPI port packs the 16-bit word to the lower 32 bits of the RXSPI buffer while the upper bits in the register are zeros.

32-bit word. No packing of the RXSPI or TXSPI registers is necessary as the entire 32-bit register is used for the data word.

Core Buffer Status

For core access to SPI, master and slave mode operates different:

1. If core access to a SPI slave is unable to keep up with the transmit/receive stream during a transfer operation (because of an interrupt or any other reason) the SPI operates according to the states of the SENDZ and GM bits in the SPICTL_x register.
 - If SENDZ = 1 and the transmit buffer is empty, the device repeatedly transmits zeros on the MOSI pin. One word is transmitted for each new transfer initiate command.
 - If SENDZ = 0 and the transmit buffer is empty, the device repeatedly transmits the last word transmitted before the transmit buffer became empty.
 - If GM = 1 and the receive buffer is full, the device continues to receive new data from the MISO pin, overwriting the older data in the RXSPI buffer.

Data Transfers

- If $GM = 0$ and the receive buffer is full, the incoming data is discarded, and the RXSPI register is not updated.
- 2. If core access to a SPI master is unable to keep up with the transmit/receive stream during a transfer operation (because of an interrupt or another reason) the SPI stalls the SPICLK until new data is read/written into the TXSPI/RXSPI buffers. In this scenario the ROVF/TUVF condition bits are set indicating an exception in the data stream.

DMA Buffer Status

If the DMA engine is unable to keep up with the transmit/receive stream during a transfer operation because of latency caused by using multiple DMA channels, the SPI operates according to the states of the SENDZ and GM bits in the SPICTLx register.

- If $SENDZ = 1$ and the transmit buffer is empty, the device repeatedly transmits zeros on the MOSI pin. One word is transmitted for each new transfer initiate command.
- If $SENDZ = 0$ and the transmit buffer is empty, the device repeatedly transmits the last word transmitted before the transmit buffer became empty.
- If $GM = 1$ and the receive buffer is full, the device continues to receive new data from the MISO pin, overwriting the older data in the RXSPI buffer.
- If $GM = 0$ and the receive buffer is full, the incoming data is discarded, and the RXSPI register is not updated.

Core Transfers

The RXS bit defines when the receive buffer can be read. The TXS bit defines when the transmit buffer can be filled. The end of a single word

transfer occurs when the RXS bit is set. This indicates that a new word has been received and latched into the receive buffer, RXSPI. The RXS bit is set shortly after the last sampling edge of SPICLK. There is a 4 PCLK cycle latency for a master/slave device, depending on synchronization. This is independent of the CPHASE and TIMOD bit settings and the baud rate.

Backward Compatibility

To maintain software compatibility with other SPI devices (68HC11), the SPI transfer finished bit (SPIF) is also available for polling. This bit may have slightly different behavior from that of other commercially available devices. For a slave device, SPIF is set at the same time as RXS. For a master device, SPIF is set one-half (0.5) of the SPICLK period after the last SPICLK edge, regardless of CPHASE or CLKPL. The baud rate determines when the SPIF bit is set. In general, SPIF is set after RXS, but at the lowest baud rate settings ($\text{SPIBAUD} < 4$). The SPIF bit is set before the RXS bit, and consequently before new data has been latched into the RXSPI buffer. Therefore, for $\text{SPIBAUD} = 2$ or $\text{SPIBAUD} = 3$, the processor must wait for the RXS bit to be set (after SPIF is set) before reading the RXSPI buffer. For larger SPI-BAUD settings ($\text{SPIBAUD} > 4$), RXS is set before SPIF.

DMA Transfers

The SPI ports support both master and slave mode DMA. DMA is enabled for TIMOD bit = 10.



Enable the SPI port before enabling DMA.

For master mode, a DMA transfer starts after the DMA engine is enabled. For slave mode the slave select pin (SPI_DS_I) needs to be asserted to start slave DMA operation.

Data Transfers

When enabled as a master, the DMA engine transmits or receives data as follows:

- If the SPI system is configured for transmitting, the DMA engine reads data from memory into the DMA FIFO. Data from the DMA FIFO is loaded into the TXSPI_x buffer and then into the transmit shift register. This initiates the transfer on the SPI port.
- If configured to receive, data from the RXSPI_x buffer is automatically loaded into the DMA FIFO. Then the DMA engine reads data from the DMA FIFO and writes to memory. Finally, the SPI initiates the receive transfer. The SPI generates the programmed signal pulses on SPICLK and the data is shifted out of MOSI and in from MISO simultaneously. The SPI continues sending or receiving words until the SPI DMA word count register transitions from 1 to 0.



Do not write to the TXSPI_x buffer during an active SPI transmit DMA operation because DMA data will be overwritten. Similarly, do not read from the RXSPI_x buffer during active SPI DMA receive operations. DMA Interrupts are generated based on DMA events and are configured in the SPIDMAC_x registers. In order for a transmit DMA operation to begin, the transmit buffer (TXSPI_x) must initially be empty (TXS = 0). While this is normally the case, this means that the TXSPI_x buffer should not be used for any purpose other than SPI transfers. Writing to the TXSPI_x buffer via the software sets the TXS bit.



For receive master DMA the SPICLK stops only when the RXSPI_x buffer and DMA FIFO are full (even if the DMA count is already zero). Therefore, SPICLK runs for an additional five word transfers filling junk data in the RXSPI_x buffer and DMA FIFO. The FIFOs must be flushed before a new DMA is initiated.

DMA Chaining

The serial peripheral interfaces support both single and chained DMA. However, unlike the serial ports, programs cannot insert a TCB in an active chain. [For more information, see “SPI TCB” on page 2-14.](#)

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



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

DMA Transfer Count

When the SPI is configured for receive/transmit DMA, the number of words configured in the DMA count register should match the actual data transmitted. When the SPI DMA is used, the internal DMA request is generated for a DMA count of four. In case the count is less than four, one DMA request is generated for all the bytes.

For example, when a DMA count of 16 is programmed, four DMA requests are generated (that is, four groups of four). For a DMA count of 18, five DMA requests are generated (four groups of four and one group of two). In case the SPI DMA is programmed with a value more than the actual data transmitted, some bytes may not be received by the SPI DMA due to the condition for generating the DMA request.

Interrupts

Full Duplex Operation

The SPI interface allows full-duplex operation running the DMA channel to the transmit/receive path and core access to the alternate transmit/receive path. For full-duplex operation, set `TIMOD = 10` which generates the interrupts for DMA only.

Reads from the `RXSPIX` buffer are allowed at any time during transmit DMA. Note the `TXS` bit is cleared when the `TXSPIX` buffer is read but the DMA FIFO is not available in the receive path. The receive interface cannot generate an interrupt, but the `RXS` status bits can be polled.

Writes to the `TXSPIX` buffer during an active SPI receive DMA operation are permitted. Note the `RXS` bit is cleared when the `RXSPIX` buffer is read but the DMA FIFO is not available in the transmit path. The transmit interface cannot generate an interrupt, but the `TXS` status bits can be polled.

Interrupts

The following section describes SPI operations using both the core and direct memory access (DMA). [Table 15-7](#) provides an overview of SPI interrupts.

Interrupt Sources

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.

Table 15-7. SPI Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
SPI (SPI Mode 3–0, 2 channels)	<ul style="list-style-type: none"> – DMA RX/TX done – Core RX buffer full – Core TX buffer empty – DMA multi master error – DMA under/overflow error 	<p>Internal transfer completion for core mode of operation. For DMA modes,</p> <p>Internal transfer completion when INTETC = 0.</p> <p>External transfer completion when INTETC = 1.</p>	RTI instruction	P1I, P18I

The `TIMOD` (transfer initiation and interrupt) register determines whether the interrupt is based on the `TXSPI` or `RXSPI` buffer status.



If configured to generate an interrupt when `SPIRX` is full (`TIMOD` = 00), the interrupt becomes active 1 `PCLK` cycle after the `RXS` bit is set.

During DMA driven transfers, an SPI interrupt is triggered:

1. At the completion of a single DMA transfer when `count` = 0 and `INTETC` = 0, or when the last data is transferred externally and `INTETC` = 1.
2. At the completion of a number of DMA sequences (if DMA chaining is enabled).
3. When a DMA error has occurred.

Note that 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` interrupt and the secondary SPI uses the `SPILI` interrupt. Whenever an SPI interrupt occurs (regardless of the cause), the `SPILI` or `SPIHI` interrupts are latched. To service the primary SPI port, unmask (set = 1) the `SPIHI` bit

Interrupts

(bit 12) in the IMASK register. To service the secondary SPI port, unmask (set = 1) the SPILIMSK bit (bit 19) in the LIRPTL register. For a list of these bits, see the *SHARC Processor Programming Reference*.

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. Depending upon the state of INTETC bit the interrupt can be generated when the internal count becomes zero or the external transfer is complete. Otherwise, set the INTERR bit to trigger the interrupt if one of the error conditions occurs during the transmission—for example a multimaster error (MME), transmit buffer underflow (TUNF – only if SPIRCV = 0), or receive buffer overflow (ROVF – only if SPIRCV = 1).



During core-driven transfers, the TUNF and ROVF error conditions do not generate interrupts.

When DMA is disabled, the processor core may read from the RXSPI register or write to the TXSPI data buffer. The RXSPI and TXSPI buffers are memory-mapped IOP registers. A maskable interrupt is generated when the receive buffer is not empty or the transmit buffer is not full. The TUNF and ROVF error conditions do not generate interrupts in these modes.

Multi Master Error

The SPIMME bit (1) is set when the SPI_DS_I input pin of a device that is enabled as a master is driven low by some other device in the system. This occurs in multimaster systems when another device is also trying to be the master.

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 SPI_DS_I input pin should be checked to ensure that it is high; otherwise, once SPIEN and SPIMS are set, another mode-fault error condition will immediately occur. The state of the input pin is reflected in the input slave select status bit (bit 7) in the SPIFLG register.

As a result of SPIEN and SPIMS being cleared, the SPI data and clock pin drivers (MOSI, MISO, and SPICLK) are disabled. However, the slave-select output pins revert to control by the processor flag I/O module registers. This may cause contention on the slave-select lines if these lines are still being driven by the processor.

Debug Features

The following sections provide information on features that help in debugging SPI software.

Shadow Receive Buffers

A pair of read-only (RO) shadow registers for the receive data buffers, RXSPI and RXSPIB are available for use in debugging software. These registers, RXSPI_SHADOW and RXSPIB_SHADOW, are located at different addresses

Effect Latency

from RXSPI, but their contents are identical to that of RXSPI. When RXSPI is read from core, the RXS bit is cleared (read only-to-clear) and an SPI transfer may be initiated (if TIMOD = 00). No such hardware action occurs when the shadow register is read. RXSPI_SHADOW is only accessible by the core.

Internal Loopback Mode

In this mode different types of loopback are possible since there is only one DMA channel available:

- Core receive and transmit transfers
- Transmit DMA and core receive transfers
- Core Transmit and DMA receive transfers

To loop data back from MOSI to MISO, the MISO pin is internally disconnected. The MOSI pin will contain the value being looped back. Programs should set the SPIEN, SPIMS, and ILPBK bits in the SPICTLX register.



Loopback operation is only used in master mode.

Loop Back Routing

The SPI supports an internal loop back mode using the SRU. [For more information, see “Loop Back Routing” on page 9-40.](#)

Effect Latency

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

Write Effect Latency

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

SPI Effect Latency

After the SPI registers are configured the effect latency is 2 PCLK cycles to enable and 2 PCLK cycles to disable.

Programming Model

The section describes which sequences of software steps are required to get the peripheral working successfully.

Changing SPI Configuration

Programs should take the following precautions when changing SPI configurations.

- The SPI configuration must not be changed during a data transfer.
- Change the clock polarity only when no slaves are selected.
- Change the SPI configuration when SPIEN = 0. For example, if operating as a master in a multislave system, and there are slaves that require different data or clock formats, then the master SPI should be disabled, reconfigured, and then re-enabled.

However, when an SPI communication link consists of:

1. A single master and a single slave,
2. CPHASE = 1 and AUTOSDS = 0 for Master, CPHASE = 1 for slave
3. The slave's slave select input is tied low

Programming Model

Then the program can change the SPI configuration. In this case, the slave is always selected. Data corruption can be avoided by enabling the slave only after configuring both the master and slave devices.

Master Mode Transfers

For core or DMA transfers, when the SPI is configured as a master, the ports should be configured and transfers started using the following steps:

1. Route all required signals (MOSI, MISO, SPICLK) for master mode including the SPI_FLGx_0 as slave select outputs.
2. Before enabling the SPI port, programs should specify which of the slave-select signals (DPI pins) to use, setting one or more of the required SPI flag select bits (DSxEN) in the SPIFLGx register. For DMA operation set TIMOD = 10.
3. Set AUTOSDS bit to 1, to ensure the slave-selects are automatically controlled by the SPI port. (When AUTOSDS = 0, only the CPHASE = 0 setting has automated control as with previous SHARC processors)
4. Write to the SPICTLx register and set the SPIMS bit to enable the device as a master. Configure the SPIBAUDx registers, and configuring the appropriate word length, transfer format, baud rate, and other necessary information.

The next steps are dependant on whether the access is a core or a DMA access.

Core Master Transfers

When a device is to be used as a master, configure the ports using the following procedure.

1. Initiate the SPI transfer by writing or reading to/from SPI buffers. The trigger mechanism for starting the transfer is dependant upon the `TIMOD` bits in the `SPICTLx` registers. See [Table 15-6 on page 15-13](#) for more details.
2. The SPI generates the programmed clock pulses on `SPICLK`. The data is shifted out of `MOSI` and shifted in from `MISO` simultaneously. Before starting to shift, the transmit shift register is loaded with the contents of the `TXSPIx` registers. At the end of the transfer, the contents of the receive shift register are loaded into the `RXSPI` buffer.
3. With each new buffer access, the SPI continues to send and receive words, according to the SPI transfer mode (`TIMOD` bit in `SPICTLx` registers). See [Table 15-6 on page 15-13](#) for more details.
4. If there are no further SPI buffer accesses the `SPICLK` signal is stalled until new core requests are received.

DMA Master Transfers

To configure the SPI port for master mode DMA transfers:

1. Define DMA receive (or transmit) transfer parameters by writing to the `IISPIx`, `IMSPIx`, and `CSPIx` registers.
2. Write to the `SPIDMACx` register to enable the SPI DMA engine (`SPIDEN`, bit 0). And configure the following:
 - A receive access (`SPIRCV = 1`) or
 - A transmit access (`SPIRCV = 0`)

Slave Mode Transfers

When the SPI is configured as a master, regardless of core or DMA the SPI ports should be configured and transfers started using the following steps.

1. Route all required signals (MOSI, MISO, SPICLK) for slave mode including the SPI_DS_I as slave select input.
2. Write to the SPICTLx and keep the (SPIMS) cleared, enabling the device as a slave and configuring the SPI system by specifying the appropriate word length, transfer format and other necessary information. For DMA operation set TIMOD = 10.

The next steps are dependant on whether the access is a core or a DMA access.

Core Slave Transfers

The following steps illustrate SPI operation in slave mode.

1. Write the data to be transmitted into the TXSPIx buffer to prepare for the data transfer.
2. When a device is enabled as a slave, the start of a transfer is triggered by a transition of the SPI_DS_I select signal to the active state (low) or by the first active edge of the clock (SPICLK), depending on the state of CPHASE.
3. The reception or transmission continues until SPI_DS_I is released or until the slave has received the proper number of clock cycles.
4. The slave device continues to receive or transmit with each new falling-edge transition on SPI_DS_I or active SPICLK clock edge.

DMA Slave Transfers

To configure the SPI port for slave mode DMA transfers:

1. Define DMA receive (or transmit) transfer parameters by writing to the `IISPIx`, `IMSPIx`, and `CSPIx` registers.
2. Write to the `SPIDMACx` register to enable the SPI DMA engine (`SPIDEN`, bit 0). And configure the following:
 - A receive access (`SPIRCV = 1`) or
 - A transmit access (`SPIRCV = 0`)

Chained DMA Transfers

The sequence for setting up and starting a chained DMA is outlined in the following steps.

1. Clear the chain pointer register.
2. Configure the TCB associated with each DMA in the chain except for the first DMA in the chain.
3. Write the first three parameters for the initial DMA to the `IISPI`, `IMSPI`, `CSPI`, `IISPIB`, `IMSPIB`, and `CSPIB` registers directly.
4. Configure the DMA settings for the entire sequence, enabling DMA and DMA chaining in the `SPIDMAC` register.
5. Begin the DMA by writing the address of a TCB (describing the second DMA in the chain) to the `CPSPI`, `CPSPB` registers.

Stopping SPI Transfers

External transfer completion is indicated by the SPI status bit `SPIFE`. For core-driven transfers it shows that the read transfer (`TIMOD = 00`) or write transfer (`TIMOD = 01`) has been completed on the external interface. For

Programming Model

receive DMA the status bit is asserted when the DMA count becomes zero. For transmit DMA the SPIFE goes high when:

- the DMA count becomes zero and
- the DMA FIFO becomes empty and
- the SPITX buffer becomes empty (TXS bit high) and
- transfer is complete (SPIF bit goes high)

Note that the SPIFE bit can go high between two DMA blocks of a chained DMA.

Switching From Transmit to a New DMA

The following sequence details the steps for switching from transmit to transmit/receive DMA.

With SPI disabled:

1. Poll the SPIFE bit in the SPISTAT register. If this bit is high the SPI can be disabled.
2. Clear the SPICTL_x register to disable the SPI. Disabling the SPI also clears the RXSPI_x/TXSPI_x buffer and the buffer status.
3. Disable DMA by clearing the SPIDMAC_x register.
4. Clear all errors by writing to the W1C-type bits in the SPISTAT_x registers. This ensures that no interrupts occur due to errors from a previous DMA operation.
5. Reconfigure the SPICTL_x register and enable the SPI ports.
6. Configure DMA by writing to the DMA parameter registers and the SPIDMAC_x registers using the SPIDEN bit (bit 0).

With enabled SPI:

1. Poll the SPIFE bit in the SPISTAT register. If this bit is high the SPI buffer can be cleared.
2. Clear the RXSPIx/TXSPIx buffers and the buffer status without disabling the SPI. This can be done by ORing 0xC0000 with the present value in the SPICTLx register. For example, programs can use the RXFLSH and TXFLSH bits to clear TXSPIx/RXSPIx and the buffer status.
3. Clear the SPIDMAC register.
4. Clear all errors by writing to the W1C-type bits in the SPISTAT register. This ensures that no interrupts occur due to errors from a previous DMA operation.
5. Reconfigure the SPICTL register to remove the clear condition on the TXSPI/RXSPI registers.
6. Configure DMA by writing to the DMA parameter registers and the SPIDMACx registers using the SPIDEN bit (bit 0).

Switching From Receive to a New DMA

Use the following sequence to switch from receive to transmit DMA. Note that TXSPIx and RXSPIx are registers but they may not contain any bits, only address information.

With disabled SPI:

1. Poll the SPIFE bit in the SPISTAT register. If this bit =1 the SPI can be disabled.
2. Clear the SPICTLx registers to disable the SPI. Disabling the SPI also clears the RXSPIx/TXSPIx register contents and the buffer status.

Programming Model

3. Disable DMA and clear the DMA FIFO by setting the `FIFOFLSH` bit in 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.
4. Clear all errors by writing to the `SPISTATx` registers. This ensures that no interrupts occur due to errors from a previous DMA operation.
5. Reconfigure the `SPICTLx` registers and enable the SPI.
6. Configure DMA by writing to the DMA parameter registers and the `SPIDMACx` registers using the `SPIDEN` bit (bit 0).

With enabled SPI:

1. Poll the `SPIFE` bit in the `SPISTAT` register. If this bit =1 the SPI can be disabled.
2. Clear the `RXSPIx/TXSPIx` registers and the buffer status without disabling the SPI by ORing `0xC0000` with the present value in the `SPICTLx` registers. Use the `RXFLSH` (bit 19) and `TXFLSH` (bit 18) bits in the `SPICTLx` registers to clear the `RXSPIx/TXSPIx` registers and the buffer status.
3. Clear the DMA FIFO by the `FIFOFLSH` bit in the `SPIDMACx` register without changing other bits. This ensures that any data from a previous DMA operation is cleared because `SPICLK` runs for five more word transfers even after the DMA count is zero in receive DMA.



The `SPIDMACx` register bits should not be changed because as this will change the DMA direction and cause bad data to be transmitted out, even with DMA disabled.

4. 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.

5. Reconfigure the SPICTL_x registers to remove the clear condition on the TXSPI_x/RXSPI_x registers.
6. Configure DMA by writing to the DMA parameter registers and the SPIDMAC_x registers using the SPIDEN bit (bit 0).

DMA Error Interrupts

The SPIUNF and SPIOVF bits of the SPIDMAC_x registers indicate transmission errors during a DMA operation in slave mode. When one of the bits is set, an SPI interrupt occurs. The following sequence details the steps to respond to this interrupt.

With SPI disabled:

1. Disable the SPI port by writing 0x00 to the SPICTL_x registers.
2. Disable DMA and clear the DMA FIFO by FIFOFLSH bit in the SPIDMAC_x register. This ensures that any data from a previous DMA operation is cleared before configuring a new DMA operation.
3. Clear all errors by writing to the W1C-type bits in the SPISTAT_x registers. This ensures that the error bits SPIOVF and SPIUNF (in the SPIDMAC_x registers) are cleared when a new DMA is configured.
4. Reconfigure the SPICTL_x registers and enable the SPI using the SPIEN bit.
5. Configure DMA by writing to the DMA parameter registers and the SPIDMAC_x registers.

Programming Model

With SPI enabled:

1. Disable DMA and clear the DMA FIFO by `FIFOFLSH` bit in the `SPIDMACx` register. This ensures that any data from a previous DMA operation is cleared before configuring a new DMA operation.
2. Clear the `RXSPIx/TXSPIx` registers and the buffer status without disabling SPI. This can be done by ORing `0xC0000` with the present value in the `SPICTLx` registers. Use the `RXFLSH` and `TXFLSH` bits to clear the `RXSPIx/TXSPIx` registers and the buffer status.
3. Clear all errors by writing to the `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 the `SPICTL` register to remove the clear condition on the `RXSPI/TXSPI` register bits.
5. Configure DMA by writing to the DMA parameter registers and the `SPIDMACx` register.

Multi-Master Transfers

The following steps show how to implement a system with two SPI devices. Since the slaves cannot initiate transfers over the bus, the master must send frames over the `MOSI` pin. This ensures that slaves can respond to the bus by sending messages over the `MISO` pin to the bus master.

1. Slave writes message to its `MISO` pin.
2. Slave starts polling its `SPI_DS_I` pin which is currently low.
3. Message is latched by current master and decoded.
4. Master deasserts the slave select signal and clears the `SPIMS` bit to become a slave.

5. If bus requester detects the SPI_DS_I pin high, it sets the SPIMS bit to get bus mastership.
6. The master selects a slave by driving its' slave select flag pin.

Programming Model

16 PERIPHERAL TIMERS

In addition to the internal core timer, the ADSP-214xx processors contain identical 32-bit peripheral timers that can be used to interface with external devices. Each timer can be individually configured in three operation modes. The timers specifications are shown in [Table 16-1](#).

Table 16-1. Timer Specifications

Feature	Timer1–0
Connectivity	
Multiplexed Pinout	No
SRU DPI Required	Yes
SRU DPI Default Routing	Yes
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A
Access Type	
Data Buffer	No
Core Data Access	N/A
DMA Data Access	N/A
DMA Channels	N/A

Features

Table 16-1. Timer Specifications (Cont'd)

Feature	Timer1–0
DMA Chaining	N/A
Interrupt Source	Core
Boot Capable	N/A
Local Memory	No
Clock Operation	f_{PCLK}

Features

The peripheral timers have the features described below.

- Independent general-purpose timers
- Three operation modes (PWM, Width capture, external watchdog)
- Global control/status registers for synchronous operation of multiple timers
- Buffered timer registers (Period and Width) to allow changes on the fly



The core timer is controlled by system registers while the peripheral timers are controlled by memory-mapped registers.

Pin Descriptions

The timer has only one pin which acts as input or output based on the timer mode as shown in [Table 16-2](#).

Table 16-2. Peripheral Timer Pin Descriptions

Internal Node	Type	Description
TIMER1-0_I	I	Timer Signal. This input is active sampled during pulse width and period capture (width capture mode) or external event watchdog (external clock mode).
TIMER1-0_O	O	Timer Signal. This output is active driven in pulse width modulation (PWM out mode).
TIMER1-0_PBEN_O	O	Timer Pin Buffer Enable Output Signal. This output is only driven in PWM out mode.

SRU Programming

Since the timer has operation modes for input (capture and external clock mode) and output (PWM out mode), it requires bidirectional junctions. [Table 16-3](#) shows the required SRU routing.

Table 16-3. Timer DPI/SRU2 Signal Connections

Internal Node	DPI Group	SRU Register
Inputs		
TIMER1-0_I	Group A	SRU_INPUT2
Outputs		
TIMER1-0_O	Group B	
TIMER1-0_PBEN_O	Group C	

See also “[DPI Routing Capabilities](#)” on page 9-25.

Register Overview

The following sections provide brief descriptions of the primary registers used to program the timers. For information on the timer registers, see [“Peripheral Timer Registers” on page A-269](#).

Status and Control Registers (TMSTAT). The (TMSTAT) register indicates the status of both timers using a single read. The TMSTAT register also contains timer enable bits. Within TMSTAT, each timer has a pair of sticky status bits, that require a write one-to-set (TIM_XEN) or write one-to-clear (TIM_XDIS) to enable and disable the timer respectively.

Counter Registers (TMxCNT). When disabled, the timer counter retains its state. When re-enabled, the timer counter is re initialized from the period/width registers based on configuration and mode. The timer counter value should not be set directly by the software. It can be set indirectly by initializing the period or width values in the appropriate mode. The counter should only be read when the respective timer is disabled. This prevents erroneous data from being returned.

Period Registers (TMxPRD). When enabled and running, the processor writes new values to the timer period and pulse width registers. The writes are buffered and do not update the registers until the end of the current period (when the timer counter register equals the timer period register).

During the *pulse width modulation* (PWM_OUT), the period value is written into the timer period registers. Both period and width register values must be updated “on the fly” since the period and width (duty cycle) change simultaneously. To insure the period and width value concurrency, a 32-bit period buffer and a 32-bit width buffer are used.

During the *pulse width and period capture* (WDTH_CAP) mode, the period values are captured at the appropriate time. Since both the period and width registers are read-only in this mode, the existing 32-bit period and width buffers are used.

During the *external event watchdog* (EXT_CLK) mode, the period register is write-only. Therefore, the period buffer is used in this mode to insure high/low period value coherency.

Pulse Width Register (TMxW). During the pulse width modulation (PWM_OUT), the width value is written into the timer width registers. Both width and period register values must be updated “on the fly” since the period and width (duty cycle) change simultaneously. To insure period and width value concurrency, a 32-bit period buffer and a 32-bit width buffer are used.

During the pulse width and period capture (WDTH_CAP) mode, both the period and width values are captured at the appropriate time. Since both the width and period registers are read-only in this mode, the existing 32-bit period and width buffers are used.

When the processor is in EXT_CLK mode, the width register is unused.

Read-Modify-Write

The traditional read-modify-write operation to enable/disable a peripheral is different for the timers. [For more information, see “Peripheral Timer Registers” on page A-269.](#)

Clocking

The fundamental timing clock of the peripheral timers is peripheral clock/4 ($\text{PCLK}/4$).

Functional Description

Each timer has one dedicated bidirectional chip signal, TIMER_x . The two timer signals are connected to the 14 digital peripheral interface (DPI)

Functional Description

pins through the signal routing unit (SRU). The timer signal functions as an output signal in PWM_OUT mode and as an input signal in WDTH_CAP and EXT_CLK modes. To provide these functions, each timer has four, 32-bit registers shown in [Figure 16-1](#).

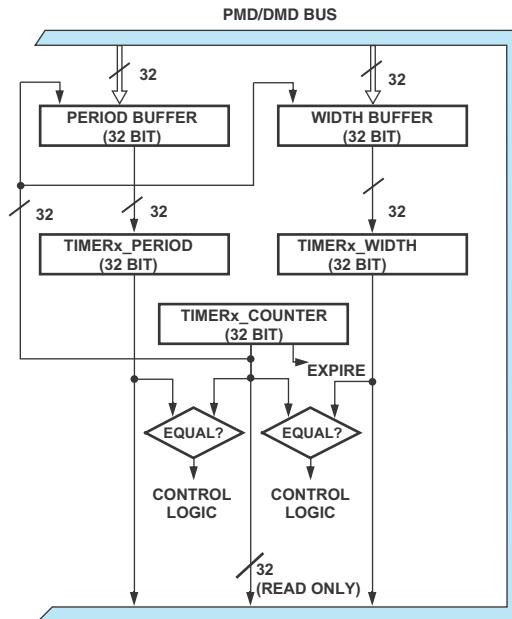


Figure 16-1. Timer Block Diagram

The registers for each timer are:

- Timer x control (TMxCTL) register
- Timer x word count (TMxCNT) register
- Timer x word period (TMxPRD) register
- Timer x word pulse width (TMxW) register

The timers also share a common status and control register—the timer global status and control (TMSTAT) register.

When clocked internally, the clock source is the processor's peripheral clock (PCLK). The timer produces a waveform with a period equal to $2 \times \text{TMxPRD}$ and a width equal to $2 \times \text{TMxW}$. The period and width are set through the TMxPRD30-0 and the TMxW30-0 bits. Bit 31 is ignored for both.

Operating Modes

The three operating modes of the peripheral timer; PWM_OUT, WDTH_CAP, and EXT_CLK, are described in [Table 16-4](#) and the following sections.

Table 16-4. Timer Signal Use

Register Settings	PWM_OUT Mode	WIDTH_CAP Mode	EXT_CLK Mode
MODE	01 = Output PWM Waveform	10 = Input Waveform	11 = Input Event
TIMEN	1 = Enable & Start Timer 0 = Disable Timer	1 = Enable & Start Timer 0 = Disable Timer	1 = Enable & Start Timer 0 = Disable Timer
PULSE	1 = Generate High Width 0 = Generate Low Width	1 = Measure High Width 0 = Measure Low Width	1 = Count at event rise 0 = Count at event fall
PRDCNT	1 = Generate PWM 0 = Single Width Pulse	1 = Measure Period 0 = Measure Width	Unused
IRQEN	1 = Enable Interrupt 0 = Disable Interrupt	1 = Enable Interrupt 0 = Disable Interrupt	1 = Enable Interrupt 0 = Disable Interrupt
Period	WO: Period Value	RO: Period Value	WO: Period Value
Width	WO: Width Value	RO: Width Value	Unused
Counter	RO: Only if not enabled Counts down on PCLK	RO: Only if not enabled Counts up on PCLK	RO: Only if not enabled Counts down on Event

Operating Modes

Table 16-4. Timer Signal Use (Cont'd)

Register Settings	PWM_OUT Mode	WIDTH_CAP Mode	EXT_CLK Mode
TMxOVF (IRQ also set)	Set if Initialized with: Period < Width or Period == Width or Period == 0	Set if the Counter wraps (Error Condition)	Unused
TMxIRQ (If enabled)	If PERIOD_CNT: 1 = Set at end of Period 0 = Set at end of Width	If PERIOD_CNT: 1 = Set at end of Period 0 = Set at end of Width	Set after Period Expires and PCLK is running

Pulse Width Modulation Mode (PWM_OUT)

In PWM_OUT mode, the timer supports on-the-fly updates of period and width values of the PWM waveform. The period and width values can be updated once every PWM waveform cycle, either within or across PWM cycle boundaries.

To enable PWM_OUT mode, set the `TIMODE1-0` bits to 01 in the timer's configuration (`TMxCTL`) register. This configures the timer's `TIMERx` signal as an output with its polarity determined by `PULSE` as follows:

- If `PULSE` is set (= 1), an active high width pulse waveform is generated at the `TIMERx` signal.
- If `PULSE` is cleared (= 0), an active low width pulse waveform is generated at the `TIMERx` signal.

The timer is actively driven as long as the `TIMODE` field remains 01.

Figure 16-2 shows a flow diagram for PWM_OUT mode. When the timer becomes enabled, the timer checks the period and width values for plausibility (independent of the value set with the `PRDCNT` bit) and does *not* start to count when any of the following conditions are true:

- Width is equal to zero

- Period value is lower than width value
- Width is equal to period

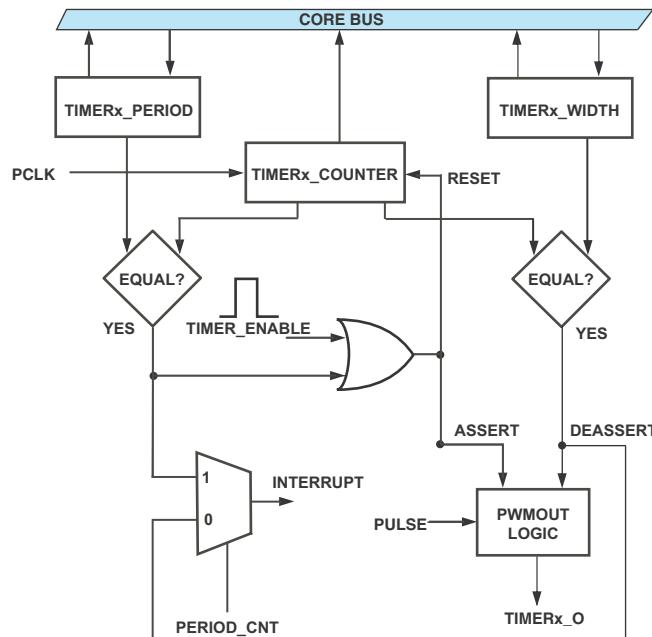


Figure 16-2. Timer Flow Diagram – PWM_OUT Mode

On invalid conditions, the timer sets both the `TIMxOVF` and the `TIMIRQx` bits and the Count register is not altered. Note that after reset, the timer registers are all zero. The PWM_OUT timing is shown in [Figure 16-3](#).

As mentioned earlier, $2 \times \text{TMxPRD}$ is the period of the PWM waveform and $2 \times \text{TMxW}$ is the width. If the period and width values are valid after the timer is enabled, the count register is loaded with the value resulting from `0xFFFF FFFF - width`. The timer counts upward to `0xFFFF FFFF`. Instead of incrementing to `0xFFFF FFFF`, the timer then reloads the counter with the value derived from `0xFFFF FFFF - (period - width)` and repeats.

Operating Modes

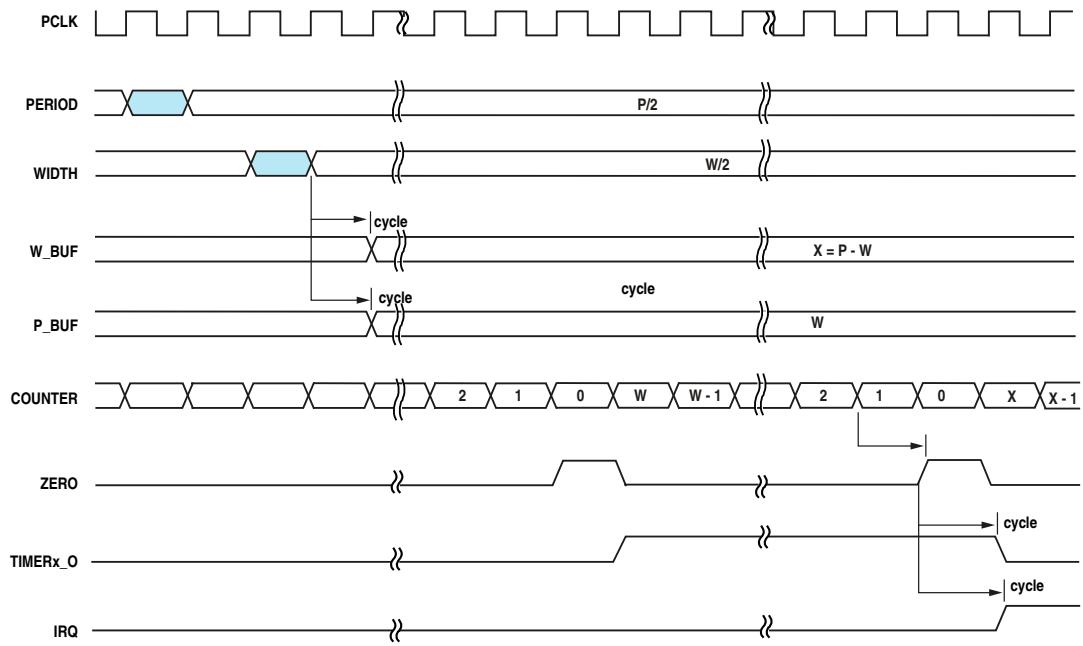


Figure 16-3. PWM_OUT Timing

PWM Waveform Generation

If the PRDCNT bit is set, the internally-clocked timer generates rectangular signals with well-defined period and duty cycles. This mode also generates periodic interrupts for real-time processing.

The 32-bit period ($\text{TM}x\text{PRD}$) and width ($\text{TM}x\text{W}$) registers are programmed with the values of the timer count period and pulse width modulated output pulse width.

When the timer is enabled in this mode, the TIMER_x signal is pulled to a deasserted state each time the pulse width expires, and the signal is asserted again when the period expires (or when the timer is started).

To control the assertion sense of the `TIMERx_0` signal, the `PULSE` bit in the corresponding `TMxCTL` register is either cleared (causes a low assertion level) or set (causes a high assertion level).

When enabled, a timer interrupt is generated at the end of each period. An ISR must clear the interrupt latch bit `TIMxIRQ` and might alter period and/or width values. In pulse width modulation applications, the program can update the period and pulse width values while the timer is running.

-  When a program updates the timer configuration, the `TMxW` register must always be written to last, even if it is necessary to update only one of the registers. When the `TMxW` value is not subject to change, the ISR reads the current value of the `TMxW` register and rewrite it again. On the next counter reload, all of the timer control registers are read by the timer.

To generate the maximum frequency on the `TIMERx_0` output signal, set the period value to two and the pulse width to one. This makes the `TIMERx` signal toggle every 2 `PCLK` clock cycles as shown in [Figure 16-9](#). Assuming `PCLK` = 133 MHz:

$$\text{Maximum period} = 2 \times (2^{31} - 1) \times 7.5 \text{ ns} = 32 \text{ seconds.}$$

-  If your application requires a more sophisticated PWM output generator, refer to [Chapter 7, Pulse Width Modulation](#).

Single-Pulse Generation

If the `PRDCNT` bit is cleared, the `PWM_OUT` mode generates a single pulse on the `TIMERx_0` signal. This mode can also be used to implement a well defined software delay that is often required by state machines. The pulse width ($= 2 \times \text{TMxW}$) is defined by the width register and the period register should be set to a value greater than the pulse width register.

At the end of the pulse, the interrupt latch bit (`TIMxIRQ`) is set and the timer is stopped automatically. If the `PULSE` bit is set, an active high pulse

Operating Modes

is generated on the `TIMERx_0` signal. If the `PULSE` bit is not set, the pulse is active low.

Pulse Mode

The waveform produced in `PWM_OUT` mode with `PRDCNT = 1` normally has a fixed assertion time and a programmable deassertion time (via the `TMxW` register). When both timers are running synchronously by the same period settings, the pulses are aligned to the asserting edge as shown in [Figure 16-4](#). Note that the timer does not support toggling of the `PULSE` bit in each period.

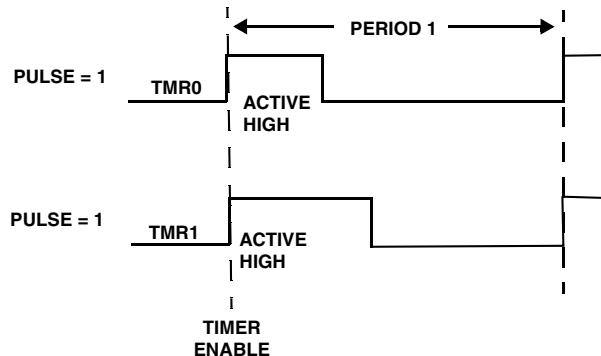


Figure 16-4. Timers with Pulses Aligned to Asserting Edge

Pulse Width Count and Capture Mode (WDTH_CAP)

To enable `WDTH_CAP` mode, set the `TIMODE1-0` bits in the `TMxCTL` register to 10. This configures the `TIMERx` signal as an input signal with its polarity determined by `PULSE`. If `PULSE` is set (= 1), an active high width pulse waveform is measured at the `TIMER_Ix` signal. If `PULSE` is cleared (= 0), an active low width pulse waveform is measured at the `TIMERx_I` signal. The internally-clocked timer is used to determine the period and pulse width of externally-applied rectangular waveforms. The period and width

registers are read-only in WDTH_CAP mode. The period and pulse width measurements are with respect to a clock frequency of $PCLK \div 2$.

Figure 16-5 shows a flow diagram for WDTH_CAP mode. In this mode, the timer resets words of the count in the $TMxCNT$ register value to 0x0000 0001 and does not start counting until it detects the leading edge on the $TIMERx_I$ signal.

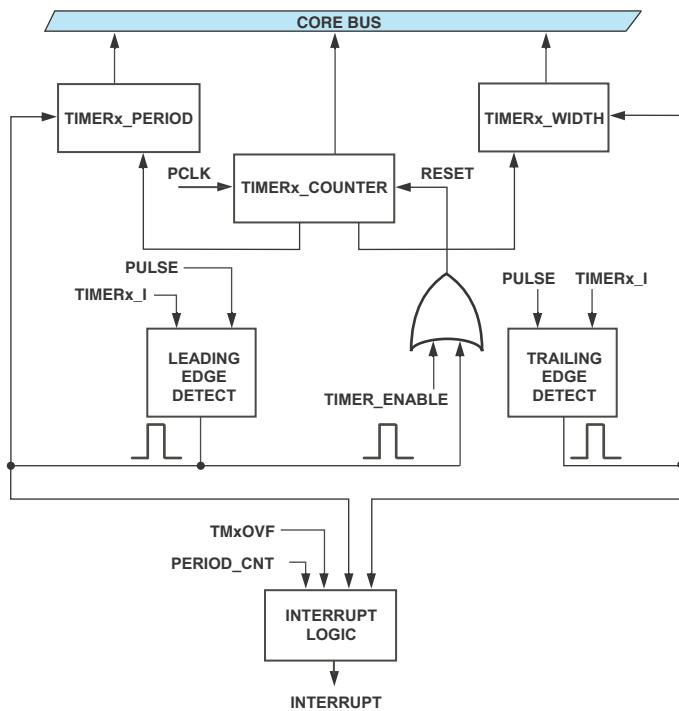


Figure 16-5. Timer Flow Diagram – WDTH_CAP Mode

When the timer detects a first leading edge, it starts incrementing. When it detects the trailing edge of a waveform, the timer captures the current value of the count register ($= TMxCNT \div 2$) and transfers it into the $TMxW$ width registers. At the next leading edge, the timer transfers the current value of the count register ($= TMxCNT \div 2$) into the $TMxPRD$ period register.

Operating Modes

The count registers are reset to 0x0000 0001 again, and the timer continues counting until it is either disabled or the count value reaches 0xFFFF FFFF.

In this mode, programs can measure both the pulse width and the pulse period of a waveform. To control the definition of the leading edge and trailing edge of the `TIMERx_I` signal, the `PULSE` bit in the `TMxCTL` register is set or cleared. If the `PULSE` bit is cleared, the measurement is initiated by a falling edge, the count register is captured to the `WIDTH` register on the rising edge, and the period register is captured on the next falling edge.

The `PRDCNT` bit in the `TMxCTL` register controls whether an enabled interrupt is generated when the pulse width or pulse period is captured. If the `PRDCNT` bit is set, the interrupt latch bit (`TIMxIRQ`) gets set when the pulse period value is captured. If the `PRDCNT` bit is cleared, the `TIMxIRQ` bit gets set when the pulse width value is captured.

If the `PRDCNT` bit is cleared, the first period value has not yet been measured when the first interrupt is generated. Therefore, the period value is not valid. If the interrupt service routine reads the period value anyway, the timer returns a period value of zero. When the period expires, the period value is loaded in the `TMxPRD` register.

A timer interrupt (if enabled) is also generated if the count register reaches a value of 0xFFFF FFFF. At that point, the timer is disabled automatically, and the `TIMxOVF` status bit is set, indicating a count overflow. The `TIMxIRQ` and `TIMxOVF` bits are sticky bits, and programs must explicitly clear them. The `WDTH_CAP` timing is shown in [Figure 16-6](#).

The first width value captured in `WDTH_CAP` mode is erroneous due to synchronizer latency. To avoid this error, programs must issue two `NOP` instructions between setting `WDTH_CAP` mode and setting `TIMxEN`.

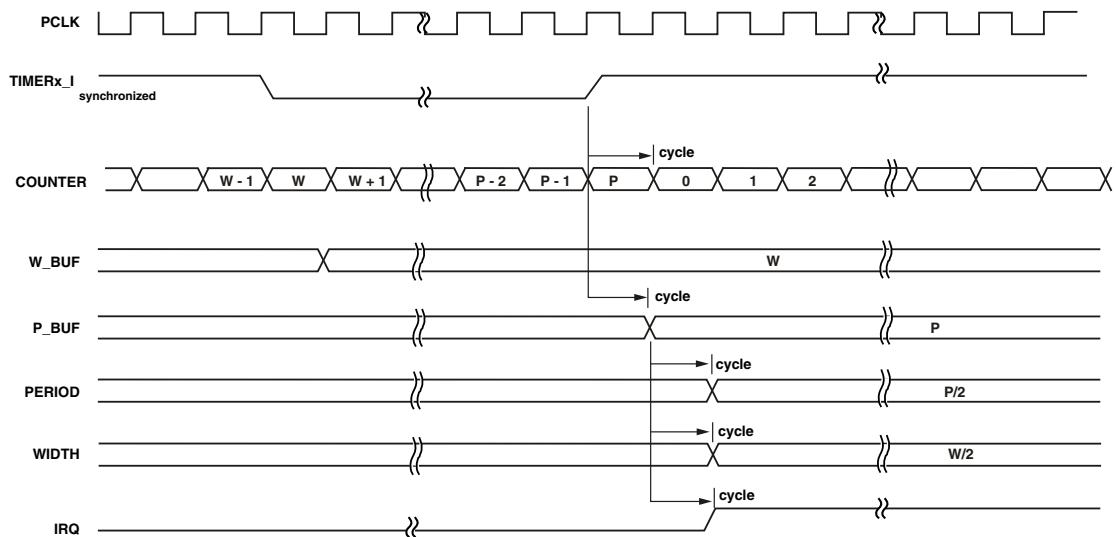


Figure 16-6. WDTH_CAP Timing (Period Count = 1)

External Event Watchdog Mode (EXT_CLK)

Figure 16-7 shows a flow diagram for EXT_CLK mode. To enable EXT_CLK mode, set the `TIMODE1-0` bits in the `TMxCTL` register to 11 in the `TMxCTL` register. This samples the `TIMERx_I` signal as an input. Therefore, in EXT_CLK mode, the `TMxCNT` register should not be read when the counter is running.

The operation of the EXT_CLK mode is as follows:

1. Program the `TMxPRD` period register with the value of the maximum timer external count.
2. Set the `TIMxEN` bits. This loads the period value in the count register and starts the countdown.
3. When the period expires, an interrupt, (`TIMxIRQ`) occurs.

Operating Modes

After the timer is enabled, it waits for the first rising edge on the `TIMERx_I` signal. The rising edge forces the count register to be loaded by the value (0xFFFF FFFF – $TMx.PRD$). Every subsequent rising edge increments the count register. After reaching the count value 0xFFFF FFFE, the `TIMx.IRQ` bit is set and an interrupt is generated. The next rising edge reloads the count register with (0xFFFF FFFF – $TMx.PRD$) again.

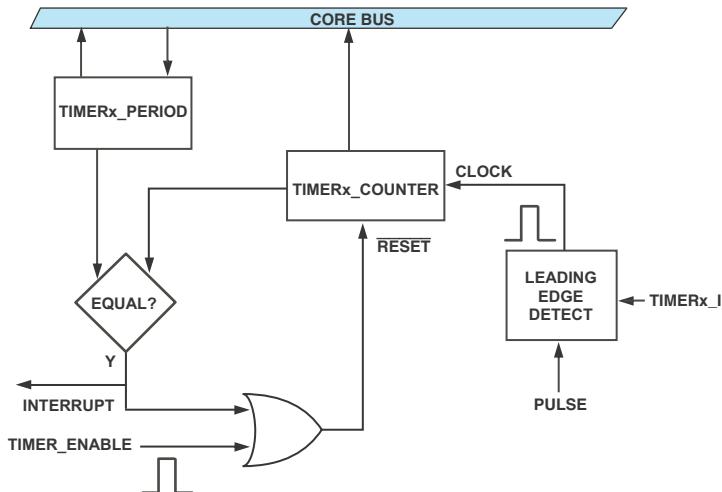


Figure 16-7. Flow Diagram EXT_CLK Mode

The EXT_CLK timing is shown in [Figure 16-8](#).

The configuration bit, `PRDCNT`, has no effect in this mode. Also, `TIMx.OVF` is never set and the width register is unused.

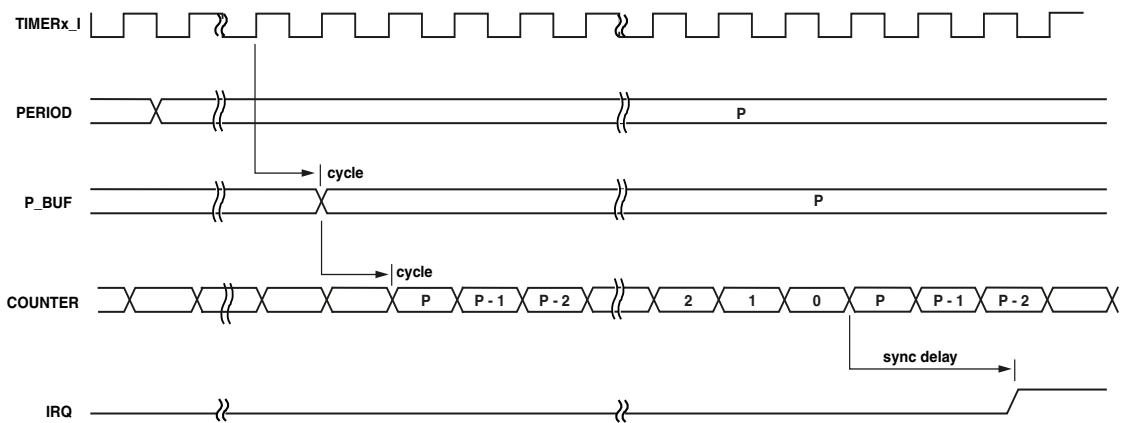


Figure 16-8. EXT_CLK Timing

Interrupts

This section describes all relevant registers and hardware to raise and service interrupts.

[Table 16-5](#) provides an overview of timer interrupts.

Table 16-5. Timer Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
GP Timer (PWM, Width Capture, Ext Watchdog, 2 channels)	– Timer expire – Timer overflow		W1C (Write 1-to-clear) TMxSTAT + RTI instruction	P2I, P10I

Sources

Each timer generates a unique interrupt request signal. A common register latches these interrupts so that a program can determine the interrupt

Interrupts

source without reference to the timer's interrupt signal. The `TMSTAT` register contains an interrupt latch bit (`TIMxIRQ`) and an overflow/error indicator bit (`TIMxOVF`) for each timer.

These sticky bits are set by the timer hardware and may be watched by software. They need to be cleared in the `TMSTAT` register by software explicitly. To clear, write a one to the corresponding bit in the `TMSTAT` register as shown in [Listing 16-1](#).

Listing 16-1. Clearing Sticky Bits

```
TMRO_ISR:  
ustat2=TIMOIRQ;  
dm(TMOSTAT)=ustat2;      /* W1C the Timer0 bit */  
r10=dm(TMOCTL);         /* dummy read for write latency */  
instructions;  
instructions;  
RTI;
```



Interrupt and overflow bits may be cleared simultaneously with timer enable or disable.

To enable a timer's interrupt, set the `IRQEN` bit in the timer's configuration (`TMXCTL`) register and unmask the timer's interrupt by setting the corresponding bit of the `IMASK` register. With the `IRQEN` bit cleared, the timer does not set its interrupt latch (`TIMxIRQ`) bits. To poll the `TIMxIRQ` bits without generating a timer interrupt, programs can set the `IRQEN` bit while leaving the timer's interrupt masked.

With interrupts enabled, ensure that the interrupt service routine (ISR) clears the `TIMxIRQ` latch before the `RTI` instruction to assure that the interrupt is not serviced erroneously. In external clock (`EXT_CLK`) mode, the latch should be reset at the very beginning of the interrupt routine so as not to miss any timer event.

Watchdog Functionality

Any of the timers can be used to implement a watchdog functionality that can be controlled by either an internal or an external clock source.

For a program to service the watchdog, the program must reset the timer value by disabling and then re-enabling the timer. Servicing the watchdog periodically prevents the count register from reaching the period value and prevents the timer interrupt from being generated. When the timer reaches the period value and generates the interrupt, reset the processor within the corresponding watchdog's ISR.

Debug Features

The following section provides information on debugging features available with the timer.

Loopback Routing

An emulation halt will not stop the timer period counter.

Loopback Routing

The timer support an internal loopback mode by using the SRU. [For more information, see “Loop Back Routing” on page 9-40.](#)

Effect Latency

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

Write Effect Latency

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

Peripheral Timers Effect Latency

After the timer registers are configured the effect latency is 3 PCLK cycles enable and 2 PCLK cycles disable. The timer starts 3 PCLK cycles after the TIMEN bit is set.

When the timer is enabled, the count register is loaded according to the operation mode specified in the TMxCTL register. When the timer is disabled, the counter registers retain their state; when the timer is re-enabled, the counter is reinitialized based on the operating mode (Figure 16-9). The program should never write the counter value directly.

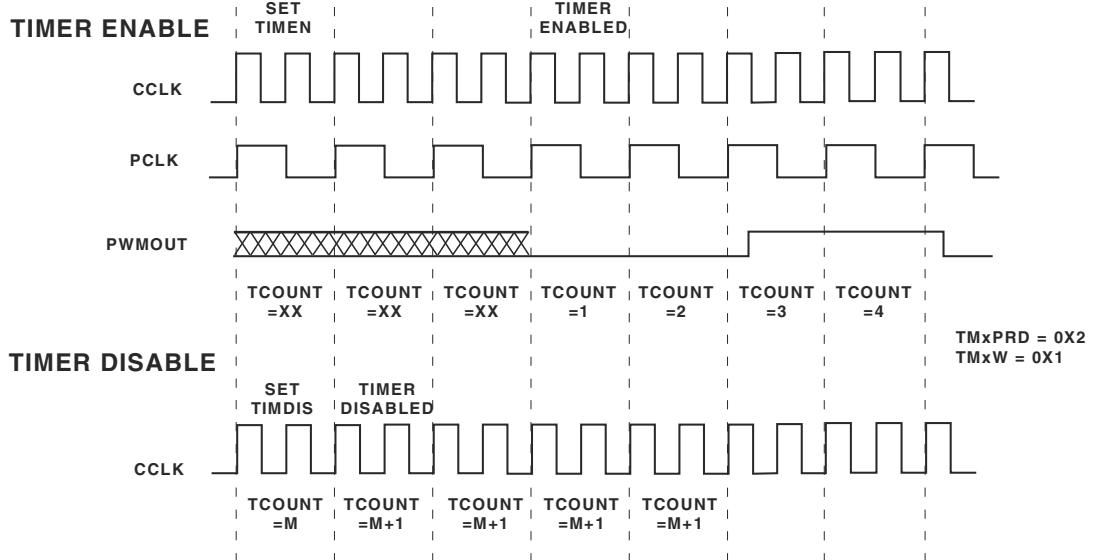


Figure 16-9. Timer PWM Enable and Disable Timing

Programming Model

The section describes which sequences of software steps are required to get the peripheral working successfully.

To enable an individual timer, set the timer's `TIMxEN` bit in the `TMSTAT` register. To disable an individual timer, set the timer's `TIMxDIS` bit in the `TMSTAT` register. To enable both timers in parallel, set all the `TIMxEN` bits in the `TMSTAT` register.

Before enabling a timer, always program the corresponding timer's configuration (`TMxCTL`) register. This register defines the timer's operating mode, the polarity of the `TIMERx` signal, and the timer's interrupt behavior. Do not alter the operating mode while the timer is running. [For more information, see “Timer Configuration Registers \(TMxCTL\)” on page A-270.](#)

PWM Out Mode

Use the following procedure to configure and run the timer in PWM out mode.

1. Reset the `TIMEN` bit and set the configuration mode to 01 to select `PWM_OUT` operation. This configures the `TIMERx_0` pin as an output pin with its polarity determined by the `PULSE` bit.
 - The timer outputs a positive active pulse width at the `TIMERx_0` pin.
 - The timer outputs a negative active pulse width at the `TIMERx_0` pin.
2. Initialize the period and width register values. Insure that the period value is greater than the width value.

Programming Model

3. Set the `TIMEN` bit. The timer performs boundary exception checks on the period and width values:
 - If (`width == 0` or `Period < width` or `period == width`) both the `OVF_ERR` and `IRQ` bits are set.
 - If there are no exceptions, the width value is loaded into the counter and it starts counting.

The timer produces PWM waveform with a period of $2 \times$ period and a width of $2 \times$ width.

- When $2 \times$ width expires, the counter is loaded with $2x(\text{period} - \text{width})$ and continues counting.
- When $2 \times$ period expires, the counter is loaded with $2 \times$ width value again and the cycle repeats.
- When the width or period expires, the `IRQ` bit (if enabled) is set depending on the `PRDCNT` bit.
- When `IRQ` is sensed, read the status register (`TMxSTAT`) and perform the appropriate “write-one” to clear.

WDTH_CAP Mode

Use the following procedure to configure and run the timer in WDTH_CAP mode.

1. Reset the `TIMEN` bit and set the configuration mode to 10 to select WDTH_CAP operation. This configures the `TIMERx_I` pin as an input pin with its polarity determined by the `PULSE` bit.
 - The timer measures a positive active pulse width at the `TIMERx_I` pin.
 - The timer measures a negative active pulse width at the `TIMERx_I` pin.

2. The PRDCNT bit determines when the $\overline{\text{TRQ}}$ status bit (if enabled) is set.
 - If ($\text{PRDCNT} == 1$), $\overline{\text{TRQ}}$ is set when the period expires and the value is captured.
 - If ($\text{PRDCNT} == 0$), $\overline{\text{TRQ}}$ is set when the width expires and the value is captured.
3. Valid period and width values are set in their respective registers when $\overline{\text{TRQ}}$ is set.

The period and width values are measured with respect to PCLK. This makes this mode coherent with the PWM_OUT mode, where the output waveforms have a period of 2 x period and a width of 2 x width.

Note that the first period value will not have been measured when the first width is measured, so it is not valid. The timer sets and returns a period value of zero in this case. When the period expires, the period value is placed into the period register. When $\overline{\text{TRQ}}$ is sensed, read the status and perform the appropriate “write-one” to clear.

EXT_CLK Mode

Use the following procedure to configure and run the timer in EXT_CLK out mode.

1. Reset the TIMEN bit and set the configuration mode to 11 to select EXT_CLK operation.

This configures the $\text{TIMER}_x\text{_I}$ pin as an input pin regardless of the setting of the PULSE bit. Note that the timer always samples the rising edge in this mode. The period register is WO and the width register is unused in this mode.

Programming Model

2. Initialize the period register with the value of the maximum external count.
3. Set the `TIMEN` bit. This loads the period value in the counter and starts the count down.

When the period expires, it is reloaded with the period value and the cycle repeats. Counter counts with each edge of the input waveform, asynchronous to `PCLK`.

When the period expires, `TRQ` (if enabled) is set and `TMR_IRQ` is asserted. An external clock can trigger the Timer to issue an interrupt and wake up an idle processor.

Reads of the count register are not supported in `EXT_CLK` mode.

17 SHIFT REGISTER – ADSP-2147X

ADSP-2147x processors incorporate an 18 stage serial in, serial/parallel out Shift Register (SR). The serial in–serial out mode can be used to delay the serial data by a fixed amount of time. The serial output can also be used to cascade the shift register modules on two or more processors. The serial in–parallel out mode can be used to convert the serial data to parallel. [Table 17-1](#) lists the shift register specifications.

Table 17-1. Shift Register Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	Yes
SRU DAI Default Routing	Yes
SRU2 DPI Required	No
SRU2 DPI Default Routing	No
Interrupt Control	N/A
Protocol	
Master Capable	N/A
Slave Capable	N/A
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A

Features

Table 17-1. Shift Register Specifications (Cont'd)

Feature	Availability
Access Type	
Data Buffer	Yes
Core Data Access	N/A
DMA Data Access	N/A
DMA Channels	N/A
DMA Chaining	N/A
Boot Capable	N/A
Local Memory	No
Clock Operation	$f_{PCLK}/4$

Features

The following list describes the features of the shift register.

- 18-stage serial/parallel shift register
- 18-bit parallel data latch
- 18 parallel output signals (`SR_LD017-0`) with can be three-stated
- Serial data input (`SR_SDI`) and output pins (`SR_SDO`) allows cascading of multiple SR registers
- SRU routing unit allows the input selection for clock and data from `SPORT7-0`, `PCGA-B`, `DAI` Pin buffer 8–1 or external SR pins
- Pin buffers remain three-stated coming out of reset until configured by software as outputs

Pin Descriptions

The pin descriptions for the shift register are described in the ADSP-2147x data sheet.

SRU Programming

To use the shift register, route the required inputs using the SRU as described in [Table 17-2](#), taking note of the following.

- The `SR_SCLK`, `SR_LAT`, and `SR_SDI` inputs must come from the same source except in the case where `SR_SCLK` comes from PCGA/B or `SR_SCLK` and `SR_LAT` come from PCGA/B. If `SR_SCLK` comes from PCGA/B then SPORT0–7 generates the `SR_LAT` and `SR_SDI` signals. If `SR_SCLK` and `SR_LAT` come from PCGA/B, then SPORT0–7 generates the `SR_SDI` signal.
- Configure `CKRE = 1` when using SPORT as a source of `SR_SCLK_I`, `SR_LAT_I`, and `SR_DAT_I` signals.
- The `SR_CTL`, `SRU_CLK_SHREG`, and `SRU_DAT_SHREG` registers are in `PCLK` domain. There may be timing violations for signals crossing `PCLK` domain to the `SR_SDCLK_I` and `SR_LAT_I` domain. To avoid this first program `SR_CTL`, `SRU_CLK_SHREG`, and `SRU_DAT_SHREG` registers and then drive on `SR_SDCLK_I`, `SR_LAT_I`, and `SR_SDI_I`.

Table 17-2. SR DAI/SRU Connections

Internal Nodes	DAI Group	SRU Register
Inputs		
<code>SR_SCLK_I</code>	G	<code>SRU_CLK_SHREG</code>
<code>SR_LAT_I</code>		
<code>SR_DAT_I</code>		<code>SRU_DAT_SHREG</code>

Register Overview

The shift register input pins (SR_CLK_I, SR_LAT_I, SR_SDI_I) are routed by default to the external shift register pins (SR_CLK, SR_LAT, SR_SDI).

Register Overview

The processor contains registers that are used to control the shift register.

- **Control Register (SHREGCTL)**. Used to clear/reset the shift register in software, select the data source for the SR_SDO_0 pin out of the 18 bits of the register, and to enable parallel data output. Complete bit description can be found at TBD.
- **Clock Routing Register (SRU_CLK_SHREG)**. Configures the clock source. [For more information, see “Clock Routing Register \(SRU_CLK_SHREG\)” on page A-145.](#)
- **Data Routing Register (SRU_DAT_SHREG)**. Configures the data source. [For more information, see “Data Routing Register \(SRU_DAT_SHREG\)” on page A-147.](#)

Clocking

The shift register requires two clock inputs: SR_SCLK_I for the serial shift register and SR_LAT_I for the latch. The source of these clocks is selectable out of many sources such as the SPORTs, PCGA/B, DAI pin buffers 8–1, or dedicated SR_SCLK and SR_LAT input pins. The data is shifted on the rising edge of the SR_SCLK_I and the data from the shift register is transferred to the latch on rising edge of the SR_LAT_I. If both clocks are connected together, the shift register is always one clock pulse ahead of the latch.

Functional Description

The Shift Register module consists of an 18-stage serial shift register, 18-bit latch, and three-state output buffers. Three-state buffers are implemented in I/O buffers. The shift register and latch have separate clocks. Data is shifted on the positive-going transitions of the `SR_SCLK_I` input. The data in each flip-flop is transferred to the respective latch on a positive-going transition of the `SR_LAT_I` input. The shift register has a serial data input (`SR_SDI_I`) and a serial data output (`SR_SDO_0`) for cascading.

A common active low asynchronous reset (`SR_CLR_I`) is provided for 18-bit shift register and for 18-bit latch. As shown in the [Figure 17-1](#), the latch has 18 parallel outputs to drive three-state output buffers. Data in the latch appears at the output whenever the output enable input (`SR_LDOE_I`) is high. The `SR_CLR_I` signal is derived from an external pin (`SR_CLR`), and a software programmable reset (`SR_CTL1`). If either of these two signals goes low, then `SR_CLR_I` goes low. The serial data output (`SR_SDO_0`) can be selected from any one of the 18-bit register's outputs. Selection of the source is provided through software using the `SR_CTL` register. A common active low asynchronous reset (`SR_CLR_I`) is provided for the shift register and for the latch.

Operating Modes

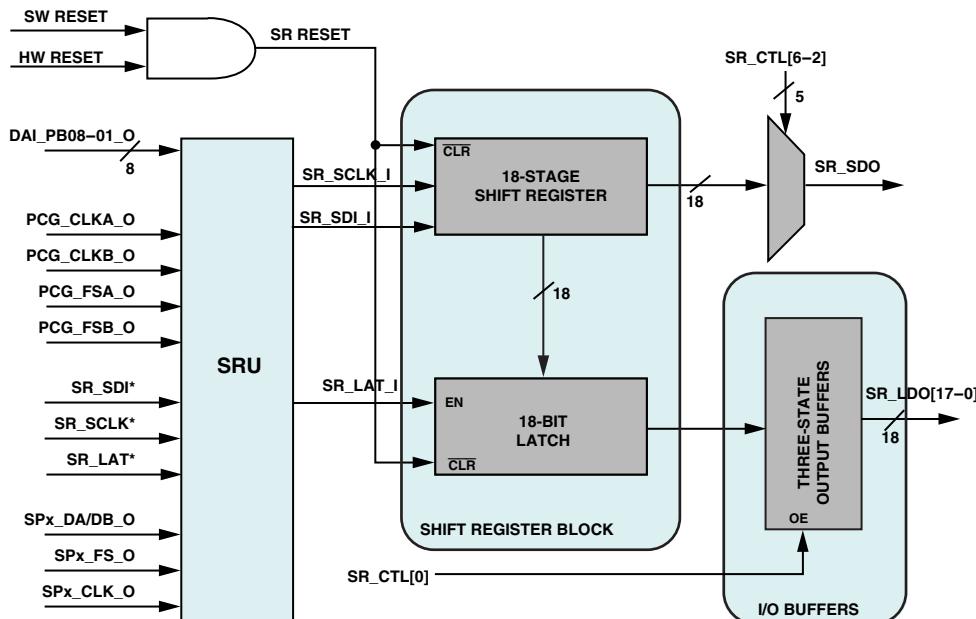


Figure 17-1. SR Block Diagram

Operating Modes

This section describes the two operation modes used by the shift register.

Serial Data Output

The shift register outputs serial data on the SR_SDO pins based on the SR_SDO_SEL bits in the SR_CTL register. These bits select which serial data of the 18-bit stream are moved to the serial output. By default if all the SR_SDO_SEL bits are cleared, the LSB data is output. This mode is useful if multiple SR registers need to be cascaded.

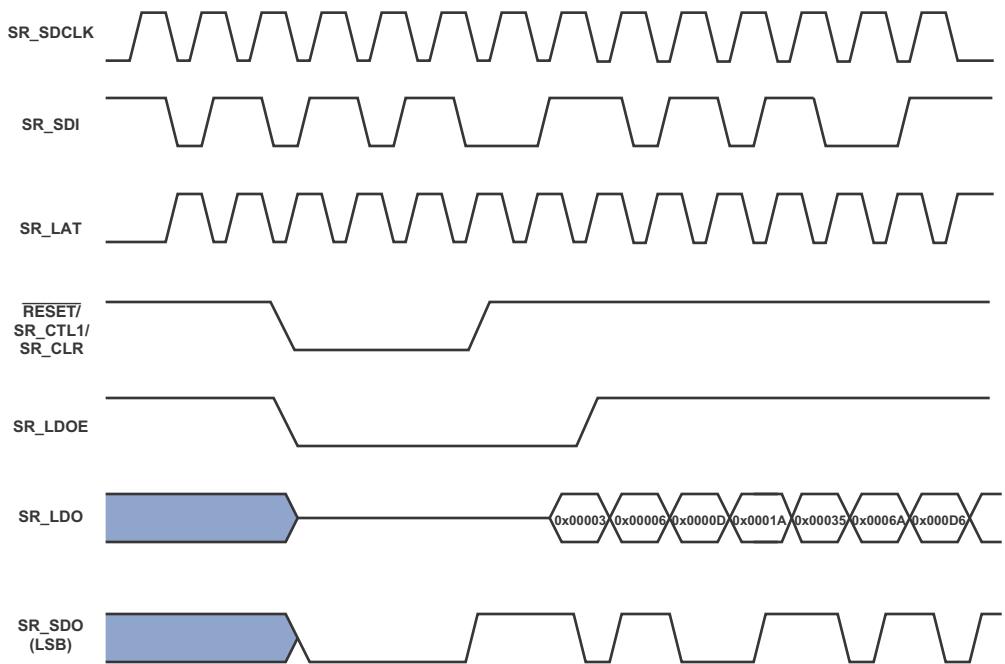


Figure 17-2. Shift Register Timing

Parallel Data Output

If the SR_LDOE bit in the SR_CTL register is set, the output stage of the parallel data latch is enabled. Data in the latch appears at the output whenever this bit is set.

The data in each flip-flop is transferred to the respective latch on a positive-going transition of the SR_LAT_I input. If both clocks are connected together, the shift register is always one clock pulse ahead of the latch.

Effect Latency

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

Write Effect Latency

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

Shift Register Effect Latency

After the SR register is configured, the maximum effect latency is 2 PCLK cycles.

Programming Model

Since the SR_CTL, SRU_CLK_SHREG, and SRU_DAT_SHREG register signals come from the peripheral clock domain (PCLK) to the SR_SDCLK_I and SR_LAT_I domain, there are timing violations for one SR_SCLK_I period. To avoid this program the following registers in the order listed.

1. The SRU_CLK_SHREG, and SRU_DAT_SHREG registers.
2. The SR_CTL register.
3. Drive the SR_SDCLK_I, SR_LAT_I, and SR_SDI_I input signals.

18 REAL-TIME CLOCK— ADSP-2147X

The ADSP-2147x processors contain a real-time clock (RTC) which provides a set of digital watch features to the processor, including time of day, alarm, and stopwatch countdown. It is typically used to implement either a real-time watch or a life counter. The RTC specifications are shown in [Table 18-1](#).

Table 18-1. RTC Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	No
SRU2 DPI Default Routing	N/A
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	No
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A
Access Type	
Data Buffer	N/A

Table 18-1. RTC Specifications (Cont'd)

Feature	Availability
Core Data Access	N/A
DMA Data Access	N/A
DMA Channels	N/A
DMA Chaining	N/A
Boot Capable	No
Local Memory	No
Clock Operation	f_{PCLK}

Features

The RTC interface has the following features.

- Provides a 1 Hz clock with Second, Minute, Hour and Day Counter (0 to 32767 days)
- Alarm Feature available with time of day interrupt
- Operates on a dedicated supply from an external 3.3 V battery
- Stopwatch function available
- Standard two pin interface with external 32.768 KHz crystal, 6 pF capacitor on each pin and 100 MΩ resistor between the pins
- RTC Power switches to that of I/O Supply when chip is powered on, saving battery life
- Calibration Feature available to correct time once a day; application can use RTCXTALIN pin to determine calibration settings

Pin Descriptions

The pins used for the real-time clock are described in the ADSP-2147x data sheet.

Clocking

The RTC timer is clocked by a 32.768 kHz crystal external to the processor. The RTC system registers are clocked by this crystal.

There is no way to disable the RTC counters from software. If a given system does not require the RTC functionality, then it may be disabled with hardware tie-offs. Tie the `RTXI` pin to `EGND`, tie the `RTCVDD` pin to `EVDD`, and leave the `RTXO` pin unconnected.

Register Overview

This section provides basic information on the RTC control and status registers only. Complete bit information can be found at “[Real-Time Clock Registers](#)” on page A-209.

Control Register (RTC_CTL). Used to control interrupts.

Clock Register (RTC_CLOCK). Used to read or write the current time and is updated every second.

Initialization Register (RTC_INIT). Used to enable the RTC. This register should only be written during initialization and not during normal operation.

Alarm Register (RTC_INIT). Used to enable the alarm functions.

Status Register (RTC_STAT). Used to report various RTC events.

Functional Description

Initialization Status Register (RTC_INITSTAT). Used to report initialization events.

Functional Description

The primary function of the RTC is to maintain an accurate day count and time of day. The RTC accomplishes this by means of four counters:

- 60-second counter
- 60-minute counter
- 24-hour counter
- 32768-day counter

The RTC increments the 60-second counter once per second and increments the other three counters when appropriate. The 32768-day counter is incremented each day at midnight (23 hours, 59 minutes, 59 seconds). Interrupts can be issued periodically, either every second, every minute, every hour, or every day. Each of these interrupts can be independently controlled. The RTC block diagram is shown in [Figure 18-1](#).

Power-up and Reset

A programmable register bit is provided to power-down the RTC. Here power-down is interpreted as a crystal oscillator disable, which would reduce power dissipation to only leakage current. Once set or reset, this bit retains its value unless changed, irrespective of the status of core supply.

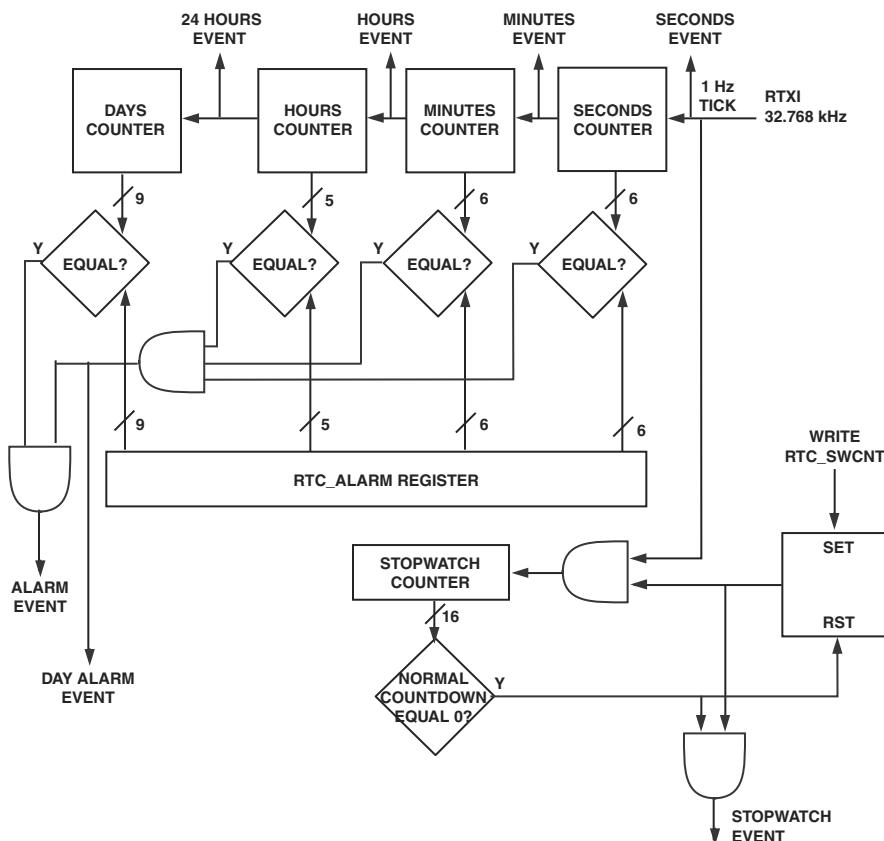


Figure 18-1. RTC Block Diagram

Digital Watch

The ADSP-2147x RTC provides a set of digital watch features. The internal oscillator generates a 32768 Hz signal using the crystal which is scaled down to 1Hz and used to clock the second, minute, hour and day counters. The 32768-day counter is incremented each day at midnight (during change from 23:59:59). The counter operates on the RTC supply (either

Functional Description

the external battery or I/O supply) and is active irrespective of the status of the processor core supply (V_{DDINT}). When the processor core and I/O supply are valid:

- the current time is updated every second into the RTC clock register (`RTC_CLOCK`)
- interrupts can be issued periodically every second, every minute, every hour or every day

Each of the interrupts can be independently controlled, described in “[Interrupts](#)” on page [18-10](#).

It is the responsibility of the program to set the correct time by a software write into the `RTC_CLOCK` register. Once set, the counters maintain time as long as the RTC supply is valid.

Partitioning

The RTC is partitioned into two blocks. The counting and clock function is provided in the I/O voltage domain (V_{DDEXT}), while the control and access function and the user interface are provided in the core voltage domain. The RTC I/O operates on a dedicated power supply provided by the external 3.3 V (nominal) lithium coin cell. The RTC also has the ability to switch to the I/O supply (V_{DDEXT}). The RTC core operates on the nominal core voltage supply (V_{DDINT}). The interface between both blocks is provided by a set of level shifters. The partitioning at chip level is shown in [Figure 18-2](#).

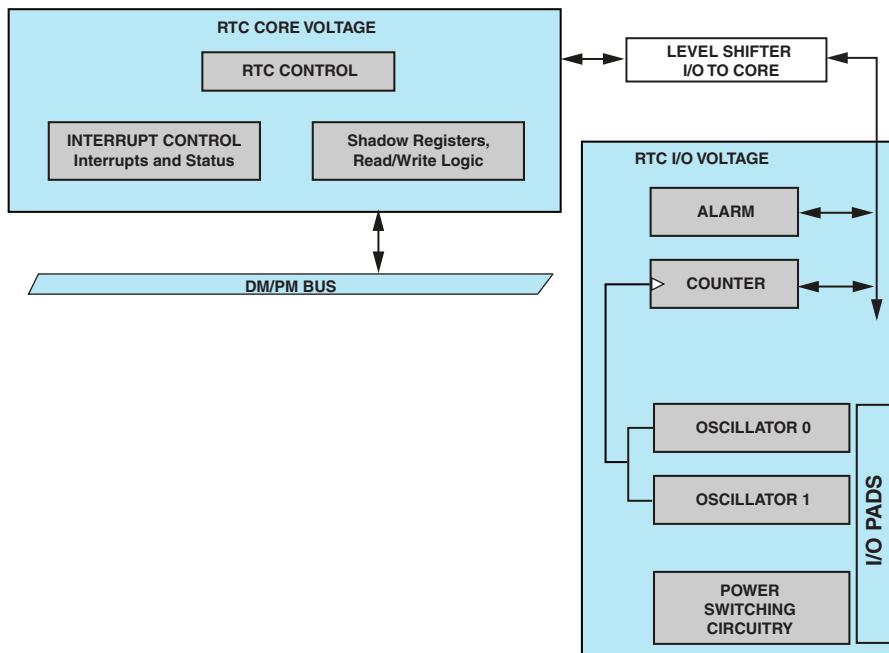


Figure 18-2. Functional Partitioning

Operating Modes

The following sections provide information on the operating modes available to the real-time clock.

Alarm

The RTC provides two alarm features, programmed with the RTC alarm register (RTC_ALARM). The first is a time of day alarm (hour, minute, and second). When the alarm interrupt is enabled, the RTC generates an interrupt each day at the time specified. The second alarm feature allows the application to specify a day as well as a time. When the day alarm

Functional Description

interrupt is enabled, the RTC generates an interrupt on the day and time specified. The alarm interrupt and day alarm interrupt can be enabled or disabled independently.

Stopwatch

The RTC provides a stopwatch function that acts as a countdown timer. The application can program a second count into the RTC Stopwatch Count register (`RTC_SWCNT`). When the stopwatch interrupt is enabled and the specified number of seconds have elapsed, the RTC generates an interrupt.

Calibration

To guard against the possibility of long term (> 1 day) errors, the RTC provides a calibration feature using 4 bits of the `RTC_INIT` register (not available for the stopwatch function).

This is a simple a time correction at the end of every day (when the clock register changes from a Day:Hour:Min:Sec value of XXX:23:59:59 to YYY:00:00:00). It functions by adding or subtracting an integer number of seconds (maximum of 7) from the start of the next day, to correct accumulated time error over the course of the previous day. The number of seconds that are added or subtracted is defined in the `RTC_INIT` register, `CALIB` field.

As an example, if there is a -50 ppm error in the 1Hz frequency, this translates into 86400×50 ppm seconds (=+4.32 seconds) error at the end of the day. That is at 00:00:00, RTC time is 4.32 seconds ahead of actual time. The RTC can correct this by adding 4 seconds (if 4 is the value written into the calibration register) to the time at 00:00:00. Therefore, from 23:59:59, the timer counter jumps directly to 00:00:04, (there is no 00:00:00 to 00:00:03 time occurrences). At the instant it jumps to 00:00:04, the error reduces to +0.32 seconds over the course of the day, which is only 3.7 ppm.

As a second example, if there is a +50 ppm error in the 1Hz frequency, this also translates into $86400 \times 50\text{ppm}$ seconds ($=-4.32$ seconds) error at the end of the day which in this case the time has to be subtracted. This is corrected in the RTC by counting 00:00:00 to 00:00:03 twice, so that the time is effectively subtracted. As soon as the RTC reaches 00:00:04, the error reduces to -0.32 seconds over the course of the previous day and accumulated error is minimized.

When the RTC is powered up for the first time, the calibration values are written once to ensure proper functionality. (If they are not to be used, write 0000). These register bits are sticky, which means that once set, they retain their value irrespective of the status of core supply.

The addition or subtraction of time can only be in integer multiples of seconds. Zero to seven seconds can be added or subtracted using 4 bits. The MSB indicates addition (0) or subtraction (1). The three LSBs indicate number of seconds (0 – 7 represented by their binary 3 bit equivalents). Because the clock runs at a time period of one second (@ 1 Hz frequency) 0.25 or 0.5 second resolution is not possible.

The calibration technique introduces a guard band for alarm by definition. In case the alarm is set within the duration of the time (Day:00:00:00 to Day:00:00:06) corrected by the calibration register, then it occurs at the nearest corrected time. This is shown in the following two examples.

- If the CALIB value in the RTC_INIT register is 0101, the RTC clock jumps from 23:59:59 to 00:00:05 due to calibration. If the alarm is set to 00:00:01, it occurs at the RTC time 00:00:05.
- If the CALIB value in the RTC_INIT register is 1101, then the RTC clock counts from 00:00:00 to 00:00:04 twice and then moves to 00:00:05. If the alarm is set to 00:00:01, it occurs at the RTC time 00:00:05.

Interrupts

In order to perform calibration on the bench, use the RTCXTALIN pin and check the ppm deviation from 32.768 KHz. This ppm error is same as in the internal 1Hz clock and the calibration register should be updated with the corresponding values as explained above.

Interrupts

The RTC has one interrupt that is programmable through the programmable interrupt priority control register (see [Appendix B, Peripheral Interrupt Control](#)). The RTCI source bit is used to connect the RTC interrupt to the peripheral interrupt inputs of the core. [Table 18-2](#) provides an overview of RTC interrupts.

Table 18-2. RTC Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
RTC Counter	-Second, minute, hour, day -daily alarm -underflow/stop-watch expiry		Read to clear RTC_STAT + RTI instruction	Need to route RTCI (PICRx) to any PxxI

Service Interrupt Sources

One interrupt is shared by all the RTC interrupt sources. The sources are based on the RTC counter:

- Per second
- Per minute
- Per hour
- Per day

- On countdown from a programmable value
- Daily at a specific time
- On a specific day and time
- On a 1 Hz clock failure
- Completion on pending writes to any 1 Hz registers

Service Interrupts

In the service routine the `RTC_STAT` register should be read to identify the cause of the interrupt. While reading the status register the RTC automatically clears the respective status bit ensuring that the cause has been cleared before ending the routine.

The RTC can be programmed to provide an interrupt at the completion of all pending writes to any of the 1 Hz registers (`RTC_CLOCK`, and `RTC_INIT`). Interrupts can be individually enabled or disabled using the RTC interrupt control register (`RTC_INIT`). Interrupt status can be determined by reading the RTC interrupt status register (`RTC_STAT`).

The RTC interrupt is set whenever an event latched into the `RTC_STAT` register is enabled in the `RTC_CTL` register. The pending RTC interrupt is cleared whenever all enabled and set bits in `RTC_STAT` are read, or when all bits in `RTC_CTL` corresponding to pending events are cleared.

Debug Features

The following section provides information on debugging features available with the watchdog timer.

Emulation Considerations

An emulation halt can optionally mask all RTC interrupts by setting the EMU_INTDIS bit in RTC_CTL register.

Effect Latency

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

Write Effect Latency

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

Real-Time Clock Effect Latency

After the RTC registers are configured the effect latency is 2 PCLK cycles.

Programming Model

Writes of the alarm, clock, stopwatch and initialization registers is performed in a two step sequence:

1. The desired values are programmed into a shadow register in the processor's core power domain and operating on the its peripheral clock (PCLK).
2. The contents of the shadow register are synchronized onto the contents of the RTC's internal clock register which operates on the 1 Hz clock in the RTC power domain.



To ensure that writes between the core voltage and RTC voltage domain are properly synchronized, all write commands should be issued immediately after a seconds' event in the `RTC_STAT` register.

This two step sequence results in a write latency of up to 1 second. While the write sequence is ongoing, the write pending (`WR_PEND`) bit is set in the `RTC_STAT` register and is cleared by hardware when the process is complete. Resetting or powering down the peripherals while a write is in progress, (that is when this bit is set) is forbidden. Subsequent writes to the same register before completion of the previous write are ignored.



Do not attempt write to the `RTC_CLOCK`, `RTC_ALARM` or `RTC_SWTCH` registers when the RTC oscillator is powered down or when the `RTC_READEN` bit is set.



During initialization, after a write of the `RTC_INIT` register, make sure that the `WR_PEND` bit is cleared before attempting writes to other registers.

Power-Up, Power-Down and Reset

The inclusion of the power-down bit (`RTCPDN`) as well as the possibility that the RTC may not be used in certain applications introduces specific constraints on the power-up and reset behavior of the RTC. These are described below.

1. When the RTC is powered-up for the first time, it remains in an undefined state until the core powers-up and the corresponding power-down bit in the `RTC_INIT` register is written by software. Programs should clear `RTCPDN` if the RTC function is desired and set if it is not.
2. After clearing the `RTCPDN` bit the application has to wait at least until the first seconds' event before it writes the timer and alarm registers. This is because the oscillator takes a `TSTARTUP` time before the clock gets generated.

Interrupts

This sequence applies only to the first time the RTC supply (battery or I/O) is connected. Once the `RTCPDN` bit is set or reset, its value is retained as long as RTC supply (battery or I/O) is valid.

3. After the RTC supply is connected for the first time and the `RTCPDN` bit has been cleared, the application is free to power-up and power-down the core supply any number of times without loss of RTC functionality (provided the RTC supply—battery or I/O, is valid). Conversely, if the `RTCPDN` bit has been set, then the RTC oscillator remains disabled irrespective of the status of the core supply.
4. The current status of the RTC power-down is updated by hardware into the initialization status register (`RTC_INITSTAT`) register. This is useful when the rest of the processor wakes up from power-down and needs to know the status of the RTC.
5. Whenever the ADSP-2147x processor core wakes up from power-down, the values of the `RTC_CLOCK`, `RTC_ALARM` and `RTC_SWTCH` registers is zero until the first seconds' event after power-up. At the first seconds' event, an arbitrary value is uploaded into these registers. To put them in a defined state software must write the desired value into these registers. In case the `RTC_CLOCK` and `RTC_ALARM` have been set before core power-down and subsequent power-up, their values are valid throughout, but can be read by the program only after the first seconds' event after power-up.

Event Flags



The unknown values in the registers at power up can cause event flags to set before the correct value is written into each of the registers. By catching the 1 Hz clock edge, the write to `RTC_CLOCK` can occur a full second before the write to `RTC_ALARM`. This would cause an extra second of delay between the validity of `RTC_CLOCK` and `RTC_ALARM`, if the value of the `RTC_ALARM` out of reset is the same as

the value written to `RTC_CLOCK`. Wait for the writes to complete on these registers before using the flags and interrupts associated with their values.

The following is a list of flags along with the conditions under which they are valid:

- Seconds (1 Hz) Event flag – Always set on the positive edge of the 1 Hz clock and after shadow registers have updated after waking from Deep Sleep. This is valid as long as the RTC 1 Hz clock is running. Use this flag or interrupt to validate the other flags.
- Write Complete – Always valid.
- Write Pending Status – Always valid.
- Minutes Event flag – Valid only after the second field in `RTC_STAT` is valid.
- Hours Event flag – Valid only after the minute field in `RTC_STAT` is valid.
- 24 Hours Event flag – Valid only after the hour field in `RTC_STAT` is valid.
- Stopwatch Event flag – Valid only after the `RTC_SWCNT` register is valid.
- Alarm Event flag – Valid only after the `RTC_STAT` and `RTC_ALARM` registers are valid.
- Day Alarm Event flag – Same as Alarm.

Interrupts

Writes posted together at the beginning of the same second take effect together at the next 1 Hz tick. The following sequence is safe and does not result in any spurious interrupts from a previous state.

1. Wait for 1 Hz tick.
2. Write new values for RTC_CLOCK, RTC_ALARM, and/or RTC_SWCNT.
3. Write 1s to clear the RTC_CLOCK flags for Alarm, Day Alarm, Stopwatch, and/or per-interval.
4. Write new value for RTC_CTL with Alarm, Day Alarm, Stopwatch, and/or per-interval interrupts enabled.
5. Wait for 1 Hz tick.
6. New values have now taken effect simultaneously.

19 WATCHDOG TIMER – ADSP-2147X

The ADSP-2147x processors include a 32-bit watchdog timer (WDT) that can be used to implement a software watchdog function. The timer can improve system reliability by forcing the processor to a known state through generation of a system reset if the timer expires before being reloaded by software. The WDT specifications are shown in [Table 19-1](#).

Table 19-1. Watchdog Timer Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DPI Required	No
SRU DPI Default Routing	No
Interrupt Control	No
Protocol	
Master Capable	N/A
Slave Capable	N/A
Transmission Simplex	N/A
Transmission Half Duplex	N/A
Transmission Full Duplex	N/A
Access Type	
Data Buffer	No
Core Data Access	N/A
DMA Data Access	N/A

Features

Table 19-1. Watchdog Timer Specifications (Cont'd)

Feature	Availability
DMA Channels	N/A
DMA Chaining	N/A
Interrupt Source	N/A
Boot Capable	N/A
Local Memory	N/A
Clock Operation	WDTCLKIN

Features

The following list provides a brief description of the watchdog timer's features.

- Programmable time out period – with about 1 second with 12 MHz clock.
- Time out resets the DSP and asserts the external reset ($\overline{\text{WDTRSTO}}$ pin). DSP is reset internally to the chip upon WDT time out.
- WDT has its own clock (WDT_CLKIN) that is independent from the SHARC CLKIN and any other clock derived from CLKIN.
- An internal oscillator to provide the clock input. This internal oscillator provides a 2 MHz (typical frequency) clock.
- Status bit available for the processor to read which is cleared on hardware reset assertion – it is not cleared on WDT generated reset.

- Programmable trip counter which allows programs to set the number of times the WDT can expire before the $\overline{\text{WDTRST0}}$ signal is asserted continuously.
- WDT space is locked and can be accessed only after unlocking the space using commands.

Pin Descriptions

The pins used for the watchdog timer are described in the ADSP-2147x data sheet.

Register Overview

The following sections provide brief descriptions of the primary registers used to program the timers. [For more information, see “Peripheral Timer Registers” on page A-269.](#)

Control Register (WDTCTL). The control register is a 32-bit system memory-mapped register used to configure the watchdog timer. Any writes made by the Software to the Register will keep it enabled. Only an External Hardware Reset can disable WDT.

Count Register (WDTCNT). Holds the 32-bit unsigned count value. The WDTCNT register must always be accessed with 32-bit read/writes.

Current Count Status Register (WDTCURCNT). Contains the current count value of the watchdog timer. Reads to WDTCURCNT return the current count value.

Status Register (WDTSTATUS). Contains the watchdog timer status information. This register is not cleared by the WDT generated reset.

Clocking

Trip Register (WDTTRIP). Sets the number of times that the WDT can expire before the `WDTRST0` pin is continually asserted until the next time hardware reset is applied.

Unlock Register (WDTUNLOCK). Protects the WDT configuration space (WDTCTL, WDTCNT, WDTCURCNT and WDTTRIP registers) against accidental writes from the processor core.

Clock Select Register (WDTCLKSEL). Selects one of the 2 clock sources for `WDTCLK`, an external source (external clock connected to `WDT_CLKIN` or a ceramic resonator connected between `WDT_CLKIN` and `WDT_CLK0`) or from internal oscillator. [For more information, see “Clocking” on page 19-4.](#)



Internal oscillator is only supported on ADSP-2147x processor models.

Clocking

The WDT provides three options for the clock source.

1. External clock source can be provided on `WDT_CLKIN` pin.
2. Ceramic resonator connected between `WDT_CLKIN` and `WDT_CLK0` pin combined with internal circuitry will generate clock. The ceramic resonator to be used is either CERALOCK CSTCR4M00G53-R0 (4 MHz) or CERALOCK CSTCC2M00G56-R0 (2 MHz).
3. An internal RC oscillator to provide the clock input. This internal RC oscillator provides a 2 MHz (typical frequency) clock.

Functional Description

The watchdog timer is used to supervise the stability of the system software. Software initializes the 32-bit count value of the timer, and then enables the timer. Thereafter, the software must reload the counter before

it counts to zero from the programmed value. This protects the system from remaining in an unknown state where software, which would normally reload the timer, has stopped running due to an external event or software error. When used in this way, software reloads the watchdog timer in a regular manner so that the downward counting timer never expires. An expiring timer then indicates that system software might be out of control.

The watchdog timer resets both the core and the internal peripherals. After an external reset, the WDT must be disabled by default. Software must be able to determine if the watchdog was the source of the hardware reset by interrogating a status bit in the watchdog timer control register.

As shown in [Figure 19-1](#) the clock source for the watchdog timer can be selected from either the internal RC oscillator or from an external oscillator.



The expired timer performs a software reset (reset core and the peripherals). [For more information, see “Processor Reset” on page 23-3.](#)

After an external reset, the WDT must be disabled by default. Software must be able to determine if the watchdog was the source of the hardware reset by interrogating a status bit in the watchdog timer control register.

Operating Mode

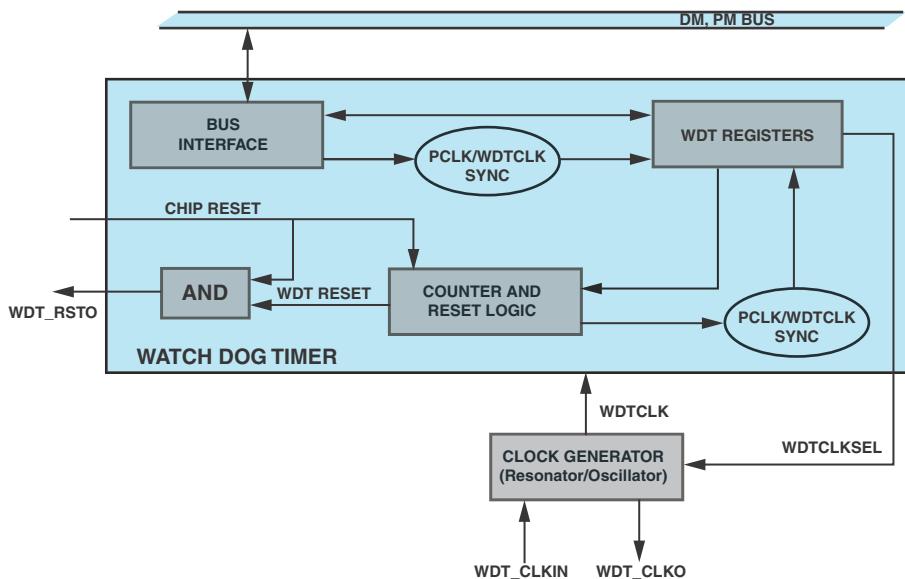


Figure 19-1. Watchdog Timer Block Diagram

Operating Mode

The WDT operates in trip count mode as described below.

Trip Count

The WDT contains a software programmable trip counter register that sets the number of times that the timer can expire before the `WDTRSTO` pin is continually asserted (until the next time hardware reset is applied). The trip counter is not cleared by the WDT generated reset. This gives software the ability to count the number of WDT generated resets using the `CURTRIPVAL` bits in the `WDTTRIP` register.

Debug Features

The following section provides information on debugging features available with the watchdog timer.

Emulation Considerations

An emulation halt stops the WDT counter. The WDT resumes counting after being released from emulation halt. Single stepping is not supported for WDT in emulation mode.

Effect Latency

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

Write Effect Latency

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

Watchdog Timers Effect Latency

After the WDT registers are configured the effect latency is 2 PCLK cycles enable and 2 PCLK cycles disable.

Programming Model

If enabled, the 32-bit watchdog timer counts downward every WDT_CLKIN cycle. When it becomes 0, the system is reset. The counter value can be read through the 32-bit WDTCURCNT register. The WDTCURCNT register cannot, however, be written directly. Rather, software writes the watchdog

Programming Model

period value into the 32-bit WDTCNT register before the watchdog is enabled. Once the watchdog is started, the period value cannot be altered.

To start the watchdog timer:

1. Unlock the WDT configuration registers by writing the unlock “command” value to the WDTUNLOCK register. Select the WDTCLK by programming the WDTCLKSEL bit.

By Default, the resonator output is WDTCLK.

2. Set the trip counter value for the watchdog timer by writing to the WDTTRIP register.
3. Set the count value for the watchdog timer by writing the count value into the watchdog period register (WDTCNT). Since the watchdog timer is not yet enabled, the write to the WDTCNT registers automatically pre-loads the WDTCURCNT register as well. Note that sufficient time must be provided for the write to the WDTCURCNT register to occur (2.5 WDTCLK cycles max.), before enabling WDT.
4. Enable the watchdog timer in WDTCTL.
5. Lock the WDT configuration registers by writing to the WDTUNLOCK register. The watchdog timer begins counting down, decrementing the value in the WDTCURCNT register every WDT_CLKIN cycle. If software does not serve the watchdog in time, WDTCURCNT continues decrementing until it reaches 0 and the system is reset.

The counter now reloads the WDTCURCNT value from WDTCNT and keeps decrementing. Additionally, the WDRO latch bit in the WDTSTATUS register is set and can be interrogated by software. This occurs up to the number of times programmed in the WDTTRIP register. When WDTTRIP expires, WDT holds the WDTRSTO asserted.

6. To prevent the watchdog from expiring, software serves the watchdog by unlocking the WDT configuration space and performing dummy writes to the WDTCURCNT register address in time. The values

written are ignored, but the write command cause the WDTCURCNT register to be reloaded from the WDTCNT register. If the watchdog is enabled with a zero value loaded to the counter, WDT expires immediately and resets the system. The WDRO bit of the watchdog control register is also set.

Programming Model

20 UART PORT CONTROLLER

The universal asynchronous receiver/transmitter (UART) is a full-duplex peripheral compatible with the PC-style, industry-standard UART. The interface specifications are shown in [Table 20-1](#).

Table 20-1. UART Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	Yes
SRU2 DPI Default Routing	Yes
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	Yes
Transmission Half Duplex	Yes
Transmission Full Duplex	Yes
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	Yes

Features

Table 20-1. UART Specifications (Cont'd)

Feature	Availability
DMA Channels	2
DMA Chaining	Yes
Boot Capable	No
Local Memory	No
Clock Operation	$f_{PCLK}/16$

Features

The UART converts data between serial and parallel formats. The serial format follows an asynchronous protocol that supports various word lengths, stop bits, and parity generation options. The UART primary features are listed below [Figure 20-1 on page 20-6](#) shows the functional block.

- Compatible with the RS-232 and RS-485 Standards
- Data packing support for efficient memory usage
- Full duplex DMA operation
- Multiprocessor communication using 9 bit addressing
- Autobaud detection support
- The UART includes interrupt handling hardware. Interrupts can be generated from 12 different events.

The UARTs do not support MODEM functionality.

SRU Programming

The SRU (signal routing unit) needs to be programmed in order to connect the UART signals to the output pins or connect the output of the transmitter to the receiver. The UART signals need to be routed as shown in [Table 20-2](#).

Table 20-2. UART Pin Descriptions

Internal Node	DPI Group	SRU Register
Input		
UART0_RX_I	Group A	SRU2_INPUT0
Outputs		
UART0_TX_O	Group A, B	
UART0_TX_PBEN_O	Group C	

Register Overview

The processor provides a set of PC-style, industry-standard control and status registers for the UART. These memory-mapped IOP registers are byte-wide registers that are mapped as half-words with the most significant byte zero-filled.

Line Control Register (UARTxLCR). Controls the format of the data character frames. It selects word length, number of stop bits and parity.

Divisor Latch High/Low Register (UARTxDLL/UARTxDLH). Characterize the UART bit rate. The divisor is split into the divisor latch low byte (UARTxDLL) and the divisor latch high byte (UARTxDLH).

Mode Control Register (UARTxMODE). Controls packing and address modes.

Clocking

Transmit Buffer Control Register (UART_xTXCTL). Controls core or DMA operation.

Receive Buffer Control Register (UART_xRXCTL). Controls core or DMA operation.

Interrupt Enable Control Register (UART_xIER). Enables interrupt requests from system handling.

Line Status Register (UART_xLSR). Returns status of controls format of the data character frames as overrun or framing errors and break interrupts.

Transmit Status Register (UART_xTXSTAT). Returns status of core or DMA operation.

Receive Status Register (UART_xRXSTAT). Returns status of core or DMA operation.

Interrupt Identification Status Register (UART_xIIR). Provides status of all interrupts and combines them into one channel.

Clocking

The fundamental timing clock of the UART module is peripheral clock/16 ($\text{PCLK}/16$).

The bit rate is characterized by the peripheral clock (PCLK) and the 16-bit divisor. The divisor is split into the UART divisor latch low byte register (UARTDLL) and the UART divisor latch high byte register (UARTDLH). These registers form a 16-bit divisor. The baud clock is divided by 16 so that:

- Divisor = 1 when $\text{UARTDLL} = 1$ $\text{UARTDLH} = 0$
- Divisor = 65,535 when $\text{UARTDLL} = \text{UARTDLH} = \text{FF}$

i The 16-bit divisor formed by the `UARTDLH` and `UARTDLL` 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 “[System and Power Management Registers](#)” on page A-4). The `UARTDLH` and `UARTDLL` registers can be programmed by software before or after turning on the clock.

[Table 20-3](#) provides example divide factors required to support most standard baud rates.

Table 20-3. UART Baud Rate Examples With 100 MHz PCLK

Baud Rate	Divisor Latch (DL)	Actual	% Error
2400	2604	2400.15	0.006
4800	1302	4800.31	0.007
9600	651	9600.61	0.006
19200	326	19,171.78	0.147
38400	163	38,343.56	0.147
57600	109	57,339.45	0.452
115200	54	115,740.74	0.469
921,600	7	892,857.14	3.119
6,250,000	1	6,250,000	–

i Careful selection of `PCLK` frequencies, that is, even multiples of desired baud rates, can result in lower error percentages.

Functional Description

The UART supports multiprocessor communication using 9-bit address detection. This allows the units to be used in multi-drop networks using

Functional Description

the RS-485 data interface standard. The UART has its own set of control and status registers (Figure 20-1).

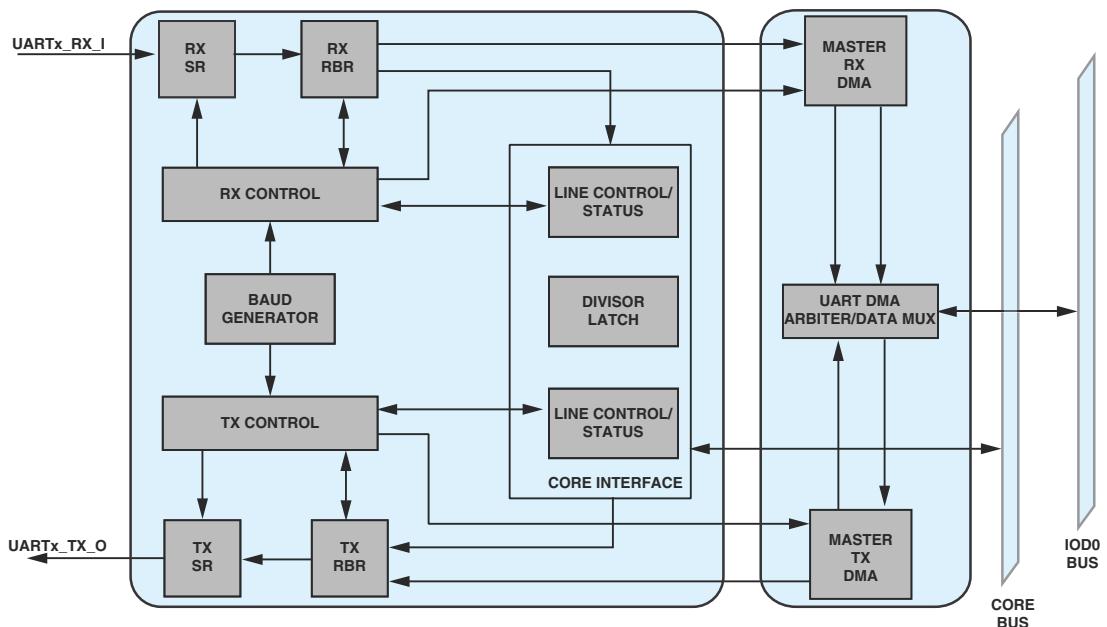


Figure 20-1. UART Functional Block Diagram

The UART is a DMA-capable peripheral with support for separate transmit and receive DMA master channels. It can be used in either DMA or core modes of operation. The core mode requires software management of the data flow using either interrupts or polling. The DMA method requires minimal software intervention as the DMA engine itself moves the data. [For more information, see “DMA Transfers” on page 20-13.](#)

Either one of the peripheral timers can be used to provide a hardware-assisted autobaud detection mechanism for use with the UART. See the [“Autobaud Detection” on page 20-21.](#)

Serial Communication

The UART follows an asynchronous serial communication protocol with these options:

- 5 – 8 data bits
- 1 or 2 stop bits
- None, even, or odd parity
- Baud rate = $\text{PCLK}/(16 \times \text{divisor})$, divisor value can be from 1 to 65,536

All data words require a start bit and at least one stop bit. With the optional parity bit, this creates a 7 to 12-bit range for each word. The format of received and transmitted character frames is controlled by the line control register (UARTLCR). Data is always transmitted and received least significant bit (LSB) first.

[Figure 20-2](#) shows a typical physical bit stream measured on the transmit pin.

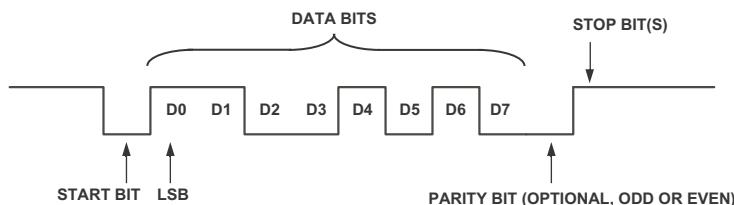


Figure 20-2. Bit Stream on the Transmit Pin

Transmit and receive channels are both buffered. The `UARTTHR` register buffers the transmit shift register (`UARTTSR`) and the `UARTRBR` register buffers the receive shift register (`UARTRSR`). The shift registers are not directly accessible by software.

Operating Modes

The two UART operation modes are described in the following sections.

Data Packing

The UART provides packed and unpacked modes of data transfer to and from the internal memory of the processors. This mode is set using the `UARTPACK` bit (bit 0) in the `UARTMODE` register. In unpacked mode, the data word is appended to the left with 24 zeros during transmission or reception. In packed mode, two words of data are transmitted or received with their corresponding higher bytes filled with zeros. For example, consecutive data words `0xAB` and `0xCD` are packed as `0x00CD 00AB` in the receiver, and `0x00CD 00AB` is transmitted as two words of `0xAB` and `0xCD` successively from the transmitter. Packing is available in both I/O and DMA modes. A control bit, `UARTPKSYN`, can be used to synchronize the packing. For information on using the UART for DMA transfers, see “[DMA Transfers](#)” on page [20-13](#).



The packed feature is provided to use the internal memory of the processor in a more efficient manner.

Note that in packed mode, both the transmitter and receiver operate with an even number of words. A transmit-buffer-empty or receive-buffer-full interrupt is generated only after an even number of words are transferred.



Programs must use care when using the packing feature in 9-bit mode.

9-Bit Transmission Mode

To select 9-bit transmission mode in the transmitter, set the `TX9` bit in the `UART_MODE` register and the `UAEN` bit in the `UART_TXCTL` register (to enable transmission). The 9-bit data (`TX9D`) can be directly written to the `UART_THR` buffer – either in packed or unpacked format. The UART

transmitter transmits the TX9D bit instead of the parity bit. During 9-bit transmission mode, the parity select controls and the word length select do not have any effect.

For the receiver, set the RX9 bit in the `UART_MODE` register and the `UAEN` bit in the `UAC_RXCTL` register (to enable reception). Set the address enable bit (`UARTAEN`) to enable address detection.

If the received ninth bit is high, the received word is shifted from the `RSR` register to the `UART_RBR` buffer which generates an interrupt. Read the `UART_RBR` buffer to find out if the device is being addressed. If the device is being addressed, the address enable (`UARTAEN` bit) is cleared to allow data and address bytes to be read into the receive buffer.

During the 9 bit transmission mode parity has to be calculated in software to detect errors. The reception may be stopped when the receiver receives another address which is different from its own.

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.

Packed Mode



Programs must use care when using the packing feature in 9-bit transmission 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 `UARTRBR` register.
- The address-detect interrupt is generated whenever the UART receiver receives an address, irrespective of the packing. The DR bit in the `UARTLSR` register can be used to discover whether the address is in the lower (`DR = 0`) or higher half-word (`DR = 1`). The LSR register must be read before reading the `UARTRBR` register, because the

Data Transfer Types

latter clears the DR bit. Reading the `UARTRBR` register clears both the address-detect and the data-ready interrupts. In non-packed mode, when the address-detect interrupt is generated, it means that the data is ready in the RBR buffer while in packed mode, this is not the case.

Data Transfer Types

The UART is capable of transferring data using both the core and DMA. Note that data packing is available using both data transfer types. [For more information, see “Data Packing” on page 20-8.](#)

Data Buffers

The UART contains a single data buffer register for transmission and reception. These buffers are described in the following sections.

Transmit Holding Registers (`UARTTHR`)

A write to the UART transmit holding register (`UARTTHR`) initiates the transmit operation. The data is moved to the internal transmit shift register (`UARTTSR`) where it is shifted out at a baud rate equal to $\text{PCLK}/(16 \times \text{Divisor})$ 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 `UARTTHR` register to the transmit shift register sets the transmit holding register empty status flag (`UARTTHRE`) in the UART line status register (`UARTLSR`).

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 20-3](#)). The `TX9DX` bits are the ninth bit in 9-bit transmission mode. A write to the UART transmit holding register

(UARTTHR) initiates the transmit operation and reads from this address return the UARTRBR register.

Note that data is transmitted and received by the least significant bit (LSB) first (bit 0) followed by the most significant bits (MSBs).

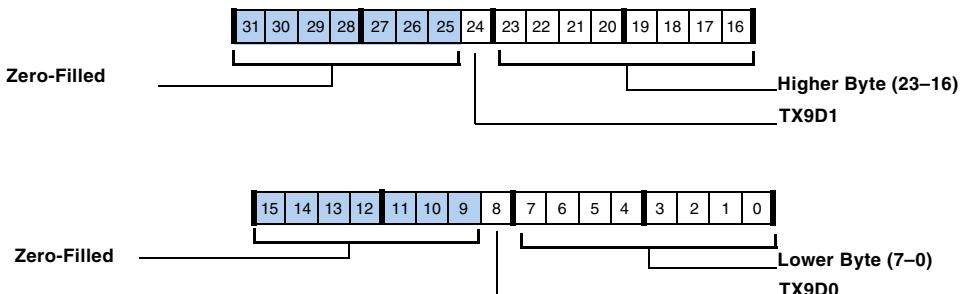


Figure 20-3. UART Transmit Holding Register (Packing Enabled)

Receive Buffer Registers (UARTRBR)

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 (UARTRSR) at a baud rate of $\text{PCLK}/(16 \times \text{Divisor})$. After the appropriate number of bits (including stop bit) is received, the data and any status are updated and the UARTRSR register is transferred to the UART receive buffer register (UARTRBR), shown in [Figure 20-4](#). After the transfer of the received word to the UARTRBR buffer and the appropriate synchronization delay, the data ready status flag (UARTDR) is updated.

A sampling clock equal to 16 times the baud rate samples the data as close to the midpoint of the bit as possible. Because the internal sample clock may not exactly match the asynchronous receive data rate, the sampling point drifts from the center of each bit. The sampling point is synchronized again with each start bit, so the error accumulates only over the

Data Transfer Types

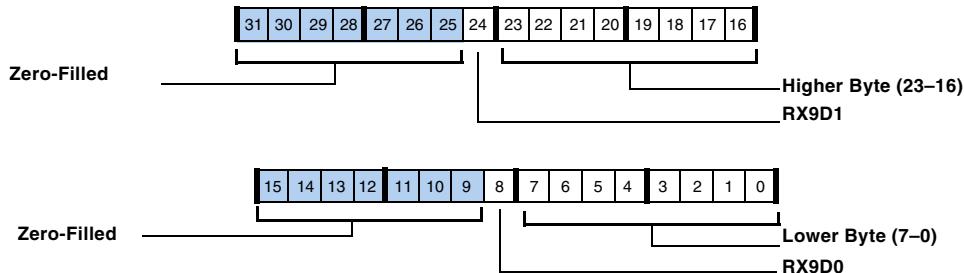


Figure 20-4. UART Receive Buffer Register

length of a single word. A receive filter removes spurious pulses of less than two times the sampling clock period.



Because of the destructive nature of reading this register, a shadow register is provided for reading the contents of the corresponding main register. [For more information, see “Debug Features” on page 20-20.](#)

Core Transfers

Core transfers move data to and from the UART by the processor core. To transmit a character, load it into the `UARTTHR` register. Received data can be read from the `UARTRBR` register. The processor must write and read one character at time.

To prevent any loss of data and misalignments of the serial data stream, the UART line status register (`UARTLSR`) provides two status flags for handshaking—`UARTTHRE` and `UARTDR`.

The `UARTTHRE` flag is set when the `UARTTHR` register is ready for new data and cleared when the processor loads new data into the `UARTTHR` register. Writing this register when it is not empty overwrites the register with the new value and the previous character is never transmitted.

The `UARTDR` flag signals when new data is available in the `UARTRBR` register. This flag is cleared automatically when the processor reads from this

register. Reading the `UARTRBR` register when it is not full returns the previously received value. When the `UARTRBR` register is not read in time, newly received data overwrites the `UARTRBR` register and the overrun (`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 can be processor-intensive, it is not typically used in real-time signal processing environments.

DMA Transfers

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

In the UART, separate receive and transmit DMA channels move data between the UART and memory. The software does not have to move data, it just has to set up the appropriate transfers either through normal DMA or DMA chaining. Software can write up to two words into the `UARTTHR` register before enabling the UART clock. As soon as the UART DMA engine is enabled, those two words are sent. See also “[Functional Description](#)” on page 2-22.

To perform DMA transfers, the UART has a special set of receive and transmit registers. These registers are listed in “[Standard DMA Parameter Registers](#)” on page 2-4.

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

DMA through the UART is started by setting up values in the DMA parameter registers and then writing to the transmit and receive control registers, enabling the module using the `UARTEN` bits (in the `UARTTXCTL` and

Interrupts

UARTRXCTL registers) and enabling DMA using the `UARTDEN` bits. A DMA can be interrupted by resetting the `UARTDEN` bit in the control register. A DMA request that is already in the pipeline completes normally.

DMA Chaining

DMA chaining is enabled by setting the `UARTCHEN` bit in the transmit and receive control registers. When chaining is enabled at the end of a current DMA, the next set of DMA parameters are loaded from internal memory and a new DMA starts. The index of the memory location is written in the chain pointer register. DMA parameter values reside in consecutive memory locations as shown in [Table 2-16 on page 2-15](#). Chaining ends when the chain pointer register contains address 0x00000 for the next parameter block.

Interrupts

The following sections provide information on the UART and interrupt generation. [Table 20-4](#) provides an overview of UART interrupts.



If UART core interrupts (core RX INT of UART) are routed via the DPI interrupt, programs do not need to read the `DPI_IRPTL` register for interrupt acknowledge. Reading the `UARTIIR` register also clears the `DPI_IRPTL` register.

Table 20-4. UART Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
DPI UART (TX/RX)	<ul style="list-style-type: none"> – DMA RX/TX done – Core RX buffer full – Core TX buffer empty – Core RX Error (overrun, parity, framing, break) – DMA RX Error (overrun, parity, framing) – Core/DMA 9-bit addressing 	Internal transfer completion	For UART DMA: Read to clear DPI_IMASK + RTI instruction For UART core (RX INT): Read to clear UARTxIIR + RTI instruction	P14I
UART (TX/RX)	<ul style="list-style-type: none"> – DMA RX/TX done – Core RX buffer full – Core TX buffer empty – Core RX Error (overrun, parity, framing, break) – DMA RX Error (overrun, parity, framing) – Core/DMA 9-bit addressing 	Internal transfer completion	For UART DMA: RTI instruction For UART core (RX INT): Read to clear UARTxIIR + RTI instruction	Need to route UARTTXI UARTRXI (PICRx) to any PxxI

Interrupt Routing

The following sections describe the various possibilities for routing a UART interrupt to the interrupt vector table (IVT).

DPI

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. Listing 20-1 shows an example of how to enable the UART over the DPI.

Interrupts

Listing 20-1. Enabling DPI UART Interrupts

```
bit set model IRPTEN;          /* enables global interrupts */
bit set imask P1I;            /* unmasks P1I interrupt */
ustat1=dm(PICR0);           /* route UART0_RXI 0x13 to P1I */
bit set ustat1 P1I4|P1I1|P1I0;
bit clr ustat1 P1I3|P1I2;
dm(PICR0)=ustat1;
```

UART

The UART receive and transmit interrupts can also be programmed through the peripheral interrupt control registers (PICRx) as separate interrupts. (By default, these interrupts are not configured in the IRPTL register—the PICRx register has to be programmed to configure them.) This method shown in [Listing 20-2](#) uses the PICR register with the code value of the UART_RXI or UART_TXI interrupts.

Listing 20-2. Enabling UART Interrupts

```
bit set model IRPTEN;          /* enables global interrupts */
bit set imask P1I;            /* unmasks P1I interrupt */
ustat1=dm(PICR0);           /* route UART0_RXI 0x13 to P1I */
bit set ustat1 P1I4|P1I1|P1I0;
bit clr ustat1 P1I3|P1I2;
dm(PICR0)=ustat1;
```

The following sections provide information on all of the available interrupt sources.

DMA Interrupts

With system DMA enabled, the UART uses DMA to transfer data to or from the processor. Dedicated DMA channels are available for receive and transmit operations. Line error handling can be configured completely independently from the receive/transmit setup.

For DMA, the transmit interrupt is generated when a DMA in transmit mode is complete whereas the receive interrupt is generated when a receive DMA is complete or when a receive error occurs. The `UARTRXSTAT` register reports whether the interrupt is due to DMA completion or errors.

For information on using the UART for DMA transfers, see “[DMA Transfers](#)” on page 20-13, “[Interrupts](#)” on page 9-32, and [Appendix B, Peripheral Interrupt Control](#).

Core Interrupts

The UART has two interrupt outputs referred to as the UART RX and UART TX interrupts. This is somewhat misleading in that in core mode all interrupts are grouped together as a single interrupt (`UART_RX`).

-  Even though the UART has two interrupts for receive and transmit, in core mode, all interrupts are grouped as a single receive interrupt (`UARTRXI`) only.

The UART interrupt enable register (`UARTIER`) is used to enable requests for core system handling of empty or full states of UART data registers. Unless polling is used as a means of action, the `UARTRBFIE` and/or `UARTTBIE` bits in this register are normally set.

Setting the bits of this register in core mode enables the UART to interrupt the processor for each word of data. For proper operation in this mode, system interrupts must be enabled, and appropriate interrupt handling routines must be present. For backward compatibility, the `UARTIIR` register still reflects the correct interrupt status. The transmit interrupt request is cleared by writing new data to the `UARTTHR` register or by reading the `UARTIIR` register.

-  When the `UARTTBIE` bit is set in the `UARTIER` register for core transfers, the UART module immediately issues an interrupt.

Interrupts

When initiating the transmission of a string, no special handling of the first character is required. Set the `UARTTBIE` bit (bit 1) and let the interrupt service routine (ISR) load the first character from memory and write it to the `UARTTHR` register in the normal manner. Accordingly, the `UARTTBIE` bit should be cleared if the string transmission has completed.

Alternatively, UART writes and reads can be accomplished by ISRs. Separate interrupt lines are provided for the transmit, receive, and error signals. The independent interrupts can be enabled individually by the `UARTIER` register.

For legacy reasons, the UART interrupt identification register (`UARTIIR`) still reflects the UART interrupt status (see [Table 20-4](#)). 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 using the programmable interrupt priority control registers.

Please note the special role of the `UARTIIR` register read in the case where the service routine does not want to transmit further data. If software stops transmission, it must read the `UARTIIR` register to reset the interrupt request. As long as the `UARTIIR` register reads 0x04 or 0x06 (indicating that another interrupt of higher priority is pending), the `UARTTHR` empty latch cannot be cleared by reading the `UARTIIR` 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 `UARTTHR` empty event.
2. Because of the destructive nature of reading the `UARTIIR` register, a shadow register (`UARTIIRSH`) is provided for reading the contents of the corresponding main register.

Error Interrupts

The `UARTLSIE` bit (bit 2 of the `UARTIER` register) 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 (`UARTLSR`):

- Receive overrun error (`UARTOE`)
- Receive parity error (`UARTPE`)
- Receive framing error (`UARTFE`)
- Break interrupt (`UARTBI`)

In core transfers, the receive interrupt is generated for the following cases.

- When `UARTRBR` 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 `UARTTHR` is empty
- An address detect (`UARTADI`) interrupt (for 9-bit mode)
- A transmit complete (`UARTTXFI`) interrupt

The ISRs can evaluate the status bit field within the UART interrupt identification register (`UARTIIR`) 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 unmasksed by the processor's interrupt controller. The ISRs must clear the interrupt latches explicitly.

Debug Features

The following sections describe the debugging features available on the UART.

Shadow Registers

Because of the destructive nature of reading the following registers: interrupt identification (UARTIIR), line status (UARTLSR) and read buffer (UARTRBR) shadow registers are provided for reading the contents of the corresponding main registers. The shadow registers, (UARTIIRSH), (UARTLSRSH) and (UARTRBRSH) return exactly the same contents as the main register, but without changing the register's status in any way.

Shadow Buffer

Because of the destructive nature of reading the read buffer (UARTxRBR) a shadow buffer is provided. The shadow buffer (UARTxRBRSH) returns exactly the same contents as the main buffer, but without changing the register's status in any way.

Loop Back Routing

The UART support an internal loop back mode by using the SRU. [For more information, see “Loop Back Routing” on page 9-40.](#)

Effect Latency

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

Write Effect Latency

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

UART Effect Latency

After the UART registers are configured the effect latency is 2 PCLK cycles. Note that when transmitting data the effective data on DPI pins can't be seen immediately after 2 PCLK cycles because the time which the UART takes to start driving data depends on the baud rate settings.

Programming Model

The following sections provide some programming procedures for core and DMA data transfers.

Autobaud Detection

When the baud rate of the incoming signal is not known, one of the general-purpose timers can be used in width capture mode to automatically calculate the baud rate. Do not enable the UART until the width is captured by the timer. To perform autobaud detection, use the following procedure.

1. The UART RX input signal is fed to a DPI pin buffer. This buffer is used as an input (`DPI_PBENxx_I` is low). The pin buffer output (`DPI_PBxx_0`) is routed to both inputs `UART_RX_I` and `TIMERx_I`.
2. Configure the timer in width capture mode with the timer interrupt enabled. Inside the ISR, the program should read the width of the incoming signal and disable the timer.

Programming Model

3. Send test data through the host device that can be used for calculating the baud rate of the incoming signal. A NULL character (0x00) can be used for this purpose.
4. The baud rate can be derived from the width of timer as follows:
$$\text{BAUDR} = \text{Width} \div (8 \times (\text{number of zero data bits} + 1))$$
.
5. When using a NULL character, the number of zero data bits is 8. Programs can also send some other pattern for this purpose. Once the baud rate is calculated inside the timer ISR, the UART can be programmed with the calculated baud rate.

Programming Model for DMA Transfers

The following is the general procedure for transferring data using DMA.

1. Clear the `UARTTXCTL/UARTRXCTL` register to zero.
2. Configure the UART DMA parameter registers (index, modify and count).
3. Configure the `UARTLCR`, `UARTDLL`, `UARTDLH`, `UARTIER`, `UARTSCR` and `UARTMODE` registers.
4. Enable the DMA by setting the `UARTEN` and `DMAEN` bits in the `UART-TXCTL/UARTRXCTL` registers.

Setting Up and Starting Chained DMA

To start a chain pointer DMA use the following steps.

1. Clear the chain pointer register
2. Initialize the chain pointer register with the address of the DMA descriptor table. Set the `PCI` bit if an interrupt is needed at the end of each DMA block.

3. Set up the appropriate control register to enable the UART transmitter and receiver, chain pointer, and DMA (UARTDEN, UARTEN, UARTCHEN bits). Once chain pointer DMA is enabled, the DMA engine fetches the index, modify, count, and chain pointer values from the memory address specified in the chain pointer register. Once the DMA parameters are fetched, normal DMA starts. This process is continued until the chain pointer register contains all zeros.

Notes on Using UART DMA

The following should be noted when performing DMA through the UART.

- DMA can be interrupted by resetting the `UARTDEN` bit, but none of the other control settings should be changed. If the UART is enabled again, then interrupted DMA can be resumed by resetting the `UARTDEN` bit.
- Disabling the UART by resetting the enable `UARTEN` bit flushes data in the transmit/receive buffer. Resetting the UART during a DMA operation is prohibited and leads to data loss.
- Do not disable chaining (`UARTCHEN` bit) when a chaining DMA is in progress.
- During a receive DMA, a read of the receiver buffer (`UARTRBR`) is not allowed. If needed, programs should read the receiver shadow buffer (`UARTRBRSH`).

Programming Model for Core Transfers

The following is the general procedure for transferring data using the core.

1. Clear the `UARTTXCTL/UARTRXCTL` registers to zero.
2. Configure the `UARTLCR`, `UARTDLL`, `UARTDLH`, `UARTSCR` and `UARTMODE` registers.
3. Program the `UARTIER` registers to generate interrupt when the transmit buffer is empty and / or the receive buffer is full.
4. Program the `PICR` registers to map the UART interrupt directly or Configure the `DPI_IRPTL` register to enable the UART interrupts.
5. Enable the UART by setting the `UARTEN` bit in the `UARTTXCTL/UARTRXCTL` registers.
6. Inside the ISR, check for the interrupt triggered and write to the transmit buffer in case of transmit buffer empty and read from the receive buffer in case of receive buffer fill event.

21 TWO WIRE INTERFACE CONTROLLER

The two wire interface (TWI) controller allows a device to interface to an inter-IC bus as specified by Philips. The TWI is fully compatible with the widely used I²C bus standard. It is designed with a high level of functionality and is compatible with multi-master, multi-slave bus configurations. To preserve processor bandwidth, the TWI controller can be set up with transfer initiated interrupts to service FIFO buffer data reads and writes only. Protocol related interrupts are optional. The TWI specifications are shown in [Table 21-1](#).

Table 21-1. TWI Specifications

Feature	Availability
Connectivity	
Multiplexed Pinout	No
SRU DAI Required	No
SRU DAI Default Routing	N/A
SRU2 DPI Required	Yes
SRU2 DPI Default Routing	Yes
Interrupt Control	Yes
Protocol	
Master Capable	Yes
Slave Capable	Yes
Transmission Simplex	
Transmission Half Duplex	
Transmission Full Duplex	

Features

Table 21-1. TWI Specifications (Cont'd)

Feature	Availability
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	No
DMA Channels	N/A
DMA Chaining	N/A
Boot Capable	No
Local Memory	No
Clock Operation	f_{PCLK}

Features

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 master controller includes the features described in the list that follows.

- Simultaneous master and slave operation on multiple device systems
- Support for multimaster data arbitration
- 7-bit addressing
- 100K bits/second and 400K bits/second data rates
- General call address support

- Master clock synchronization and support for clock low extension
- Separate multiple-byte receive and transmit FIFOs
- Low interrupt rate
- Individual override control of data and clock lines in the event of a bus lockup
- Input filter for spike suppression

The TWI moves 8-bit data externally while maintaining compliance with the I²C bus protocol.

Pin Descriptions

[Table 21-2](#) 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 21-2. TWI Pins

Internal Node	Type	Description
TWI_CLK_I	I	TWI Clock Signal. Serial clock input.
TWI_DATA_I	I	TWI Data Signal. Serial receive data input.
TWI_CLK_PBEN_O	O	TWI Clock Signal. This output signal is used to drive the TWI clock off chip Note since the TWI output signals must operate in open drain it should be routed to a DPI PBEN input.
TWI_DATA_PBEN_O	O	TWI Data Signal. This output signal is used to drive the TWI data off chip. Note that since the TWI output signals must operate in open drain it should be routed to a DPI PBEN input.

SRU Programming

The TWI signals are available through the SRU2, and are routed as described in [Table 21-3](#).

Table 21-3. TWI DPI/SRU2 Signal Connections

Internal Node	DPI Group	SRU2 Register
Inputs		
TWI_CLK_I TWI_DATA_I	Group A	SRU2_INPUT0
Outputs		
TWI_CLK_PBEN_O TWI_DATA_PBEN_O	Group C	

Clocking

The fundamental timing clock of the TWI module is peripheral clock (`PCLK`). Serial clock frequencies can vary from 400 kHz to less than 20 kHz. The resolution of the generated clock is 1/10 MHz or 100 ns.

$$\text{CLKDIV} = \text{TWI_CLOCK period} \div 10 \text{ MHz time reference}$$

For example, for an `TWI_CLOCK` of 400 kHz (period = 1/400 kHz = 2500 ns) and an internal time reference of 10 MHz (period = 100 ns):

$$\text{CLKDIV} = 2500 \text{ ns} \div 100 \text{ ns} = 25$$

For an `TWI_CLOCK` with a 30% duty cycle, then `CLKLOW` = 17 and `CLKHI` = 8. Note that `CLKLOW` and `CLKHI` add up to `CLKDIV`.

Register Overview

This section provides brief descriptions of the major registers. For complete information see “[Two Wire Interface Registers](#)” on page A-253.

Slave Mode Control Register (TWISCTL). Controls the logic associated with slave mode operation. Settings in this register do not affect master mode operation and should not be modified to control master mode functionality.

Slave Mode Status Register (TWISSTAT). During and at the conclusion of slave mode transfers, the TWISSTAT holds information on the current transfer. Generally, slave mode status bits are not associated with the generation of interrupts. Master mode operation does not affect slave mode status bits.

Master Mode Control Register (TWIMCTL). Controls the logic associated with master mode operation. Bits in this register do not affect slave mode operation and should not be modified to control slave mode functionality.

Master Mode Status Register (TWIMSTAT). Holds information during master mode transfers and at their conclusion. Generally, master mode status bits are not directly associated with the generation of interrupts but offer information on the current transfer. Slave mode operation does not affect master mode status bits.

Control Timer Register (TWIMITR). Enables the TWI module and establishes a relationship between the peripheral clock (PCLK) and the TWI controller’s internally-timed events. The internal time reference is derived from PCLK using the prescaled value shown below.

$$\text{PRESCALE} = f_{\text{PCLK}} / 10 \text{ MHz}$$

Serial Clock Divider Register (TWIDIV). During master mode operation, the TWIDIV register values are used to create the high and low durations of the TWI_CLOCK.

Functional Description

FIFO Control Register (TWIFIFOCTL). The FIFO control register affects only the FIFO and is not tied in any way with master or slave mode operation.

FIFO Status Register (TWIFIFOSTAT). The fields in the TWI FIFO status register indicate the state of the FIFO buffers' receive and transmit contents. The FIFO buffers do not discriminate between master data and slave data. By using the status and control bits provided, the FIFO can be managed to allow simultaneous master and slave operation.

Functional Description

[Figure 21-1](#) illustrates the overall architecture of the TWI controller.

The peripheral interface supports the transfer of 32-bit wide data and is used by the processor in the support of register and FIFO buffer reads and writes.

The register block contains all control and status bits and reflects what can be written or read as outlined by the programmer's model. Status bits can be updated by their respective functional blocks.

The FIFO buffer is configured as a 1-byte-wide, 2-deep transmit FIFO buffer and a 1-byte-wide, 2-deep receive FIFO buffer.

The transmit shift register serially shifts its data out externally off chip. The output can be controlled to generate acknowledgements or it can be manually overwritten.

The receive shift register receives its data serially from off chip. The receive shift register is 1 byte wide and data received can either be transferred to the FIFO buffer or used in an address comparison.

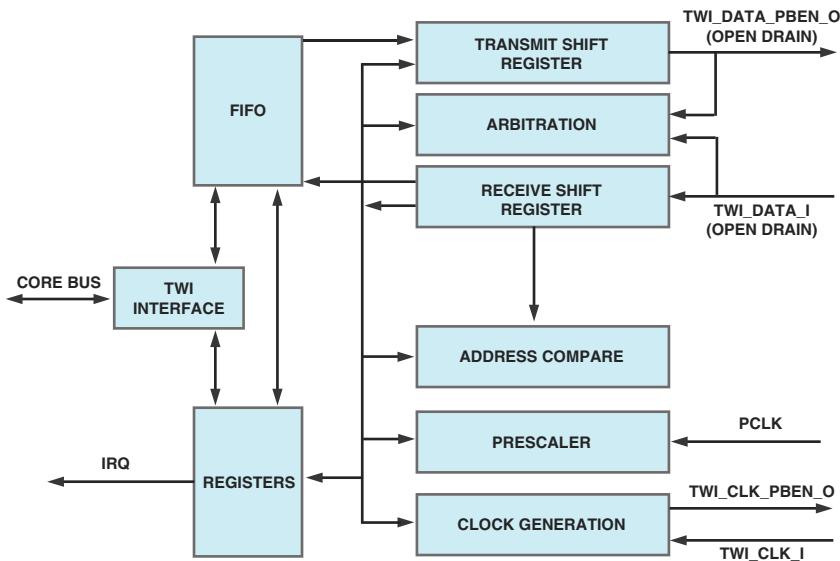


Figure 21-1. TWI Block Diagram

The address compare block supports address comparison in the event the TWI controller module is accessed as a slave.

The prescaler block must be programmed to generate a 10 MHz time reference relative to the peripheral clock. This time base is used for filtering data and timing events specified by the electrical parameters in the data sheet (see the I²C bus specification from Philips), as well as for **TWI_CLOCK** clock generation.

The clock generation module is used to generate an external serial clock (**TWI_CLOCK**) when in master mode. It includes the logic necessary for synchronization in a multimaster clock configuration and clock stretching when configured in slave mode.

Functional Description

The TWI controller's clock 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 `TWI_CLOCK` line is detected at a logic 1 level. At this time, the clock high count begins.

The TWI controller only issues a clock during master mode operation and only at the time a transfer has been initiated. If arbitration for the bus is lost, the serial clock output immediately three-states. If multiple clocks attempt to drive the serial clock line, the TWI controller synchronizes its clock with the other remaining clocks. This is illustrated in [Figure 21-2](#).

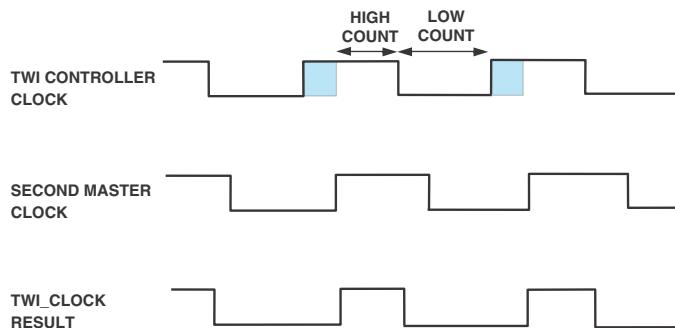


Figure 21-2. TWI Clock Synchronization

The TWI controller follows the transfer protocol of the *Philips I²C Bus Specification version 2.1* dated January 2000. A simple complete transfer is diagrammed in [Figure 21-3](#).

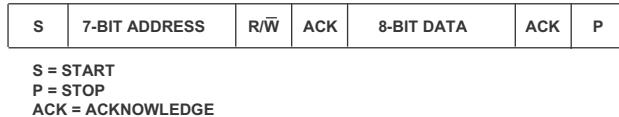


Figure 21-3. Standard Data Transfer

To better understand the mapping of TWI controller register contents to a basic transfer, [Figure 21-4](#) 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.

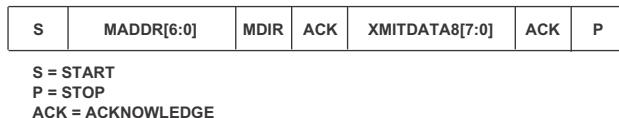


Figure 21-4. Data Transfer With Bit Illustration

Bus Arbitration

The TWI controller initiates a master mode transmission (`TWIMEN`) only when the bus is idle. If the bus is idle and two masters initiate a transfer, arbitration for the bus begins. This is illustrated in [Figure 21-5](#).

The TWI controller monitors the serial data bus (`TWI_DATA`) while the `TWI_CLOCK` is high. If `TWI_DATA` is determined to be an active logic 0 level while the internal TWI controller's data is a logic 1 level, the TWI controller has lost arbitration and ends generation of clock and data. Note that arbitration is performed not only at serial clock edges, but also during the entire time `TWI_CLOCK` is high.

Functional Description

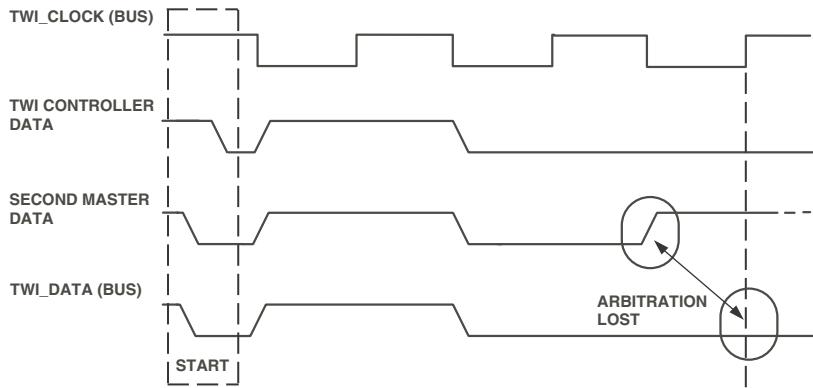


Figure 21-5. TWI Bus Arbitration

Start and Stop Conditions

Start and stop conditions involve serial data transitions while the serial clock is at logic 1 level. The TWI controller generates and recognizes these transitions. Typically, start and stop conditions occur at the beginning and at the conclusion of a transmission, with the exception of repeated start “combined” transfers, as shown in [Figure 21-6](#).

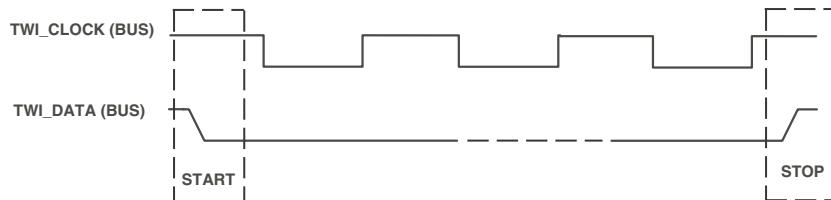


Figure 21-6. TWI Start and Stop Conditions

The TWI controller's special-case start and stop conditions include:

- TWI controller addressed as a slave-receiver
 - If the master asserts a stop condition during the data phase of a transfer, the TWI controller concludes the transfer (TWISCOMP).
- TWI controller addressed as a slave-transmitter
 - If the master asserts a stop condition during the data phase of a transfer, the TWI controller concludes the transfer (TWISCOMP) and indicates a slave transfer error (TWISERR).
- TWI controller as a master-transmitter or master-receiver
 - If the stop bit is set during an active master transfer, the TWI controller issues a stop condition as soon as possible to avoid any error conditions (as if data transfer count had been reached).

Slave Mode Addressing

With the appropriate selection of 7-bit addressing using the TWISLEN bit, the corresponding number of address bits (SADDR) are referenced during the address phase of a transfer.

Master Mode Addressing

Whether enabled as a master-transmitter or master-receiver with 7-bit addressing using the TWIMLEN bit, the TWI master performs all addressing and data transfers as required. This includes generating the repeated start condition, re-transmission of the 7-bits of the first address byte, and acknowledgement and generation of a new transfer direction change (indicated by the TWIMLEN bit).

Data Transfer

The TWI uses its transmit and receive buffers for data transfer (no DMA capability). These buffers are described in the following sections.

Data Buffers

The TWI has two data buffer FIFOs, which are described in the following sections.

8-Bit Transmit FIFO Register

The TWI 8-bit transmit FIFO register (`TXTWI8`) shown in [Figure 21-7](#) holds an 8-bit data value written into the FIFO buffer. Transmit data is entered into the corresponding transmit buffer in a first-in, first-out order. Although peripheral bus writes are 32 bits, a write access to the `TXTWI8` register adds only one transmit data byte to the FIFO buffer. With each access, the transmit status (`TWITXS`) field in the `TWIFIFOSTAT` register is updated. If an access is performed while the FIFO buffer is full, the core waits until there is at least one byte space in the transmit FIFO buffer and then completes the write access.

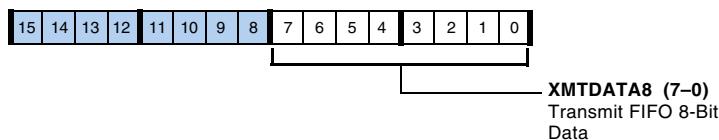


Figure 21-7. 8-Bit Transmit FIFO Register

16-Bit Transmit FIFO Register

The TWI 16-bit FIFO transmit register (`TXTWI16`) shown in [Figure 21-8](#) holds a 16-bit data value written into the FIFO buffer. Although peripheral bus writes are 32 bits, a write access to the `TXTWI16` register adds only two transmit data bytes to the FIFO buffer. To reduce interrupt output

rates and peripheral bus access times, a double byte transfer data access can be performed. Two data bytes can be written, effectively filling the transmit FIFO buffer with a single access.

The data is written in little-endian byte order where byte 0 is the first byte to be transferred and byte 1 is the second byte to be transferred. With each access, the transmit status (TWITXS) field in the TWIFIFOSTAT register is updated. If an access is performed while the FIFO buffer is not empty, the core waits until the FIFO buffer is completely empty and then completes the write access.

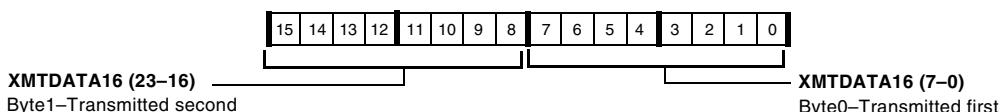


Figure 21-8. 16-Bit Transmit FIFO Register

8-Bit Receive FIFO Register

The TWI 8-bit FIFO receive register (RXTWI8) shown in [Figure 21-9](#) holds an 8-bit data value read from the FIFO buffer. Receive data is read from the corresponding receive buffer in a first-in, first-out order. Although peripheral bus reads are 32 bits, a read access to the RXTWI8 register can only access one receive data byte from the FIFO buffer. With each access, the receive status (TWIRXS) field in the TWIFIFOSTAT register is updated. If an access is performed while the FIFO buffer is empty, the core waits until there is at least one byte in the receive FIFO buffer and then completes the read access.

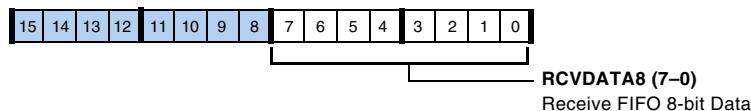


Figure 21-9. 8-Bit Receive FIFO Register

Operating Modes

16-Bit Receive FIFO Register

The TWI 16-bit FIFO receive register (RXTWI16) shown in [Figure 21-10](#) holds a 16-bit data value read from the FIFO buffer. Although peripheral bus reads are 32 bits, a read access to the RXTWI16 register can only access two receive data bytes from the FIFO buffer. To reduce interrupt output rates and peripheral bus access times, a double-byte receive data access can be performed. Two data bytes can be read, effectively emptying the receive FIFO buffer with a single access.

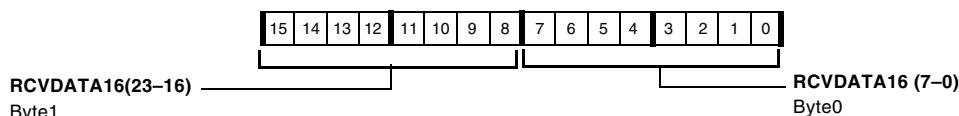


Figure 21-10. 16-Bit Receive FIFO Register

The data is read in little-endian byte order where byte 0 is the first byte received and byte 1 is the second byte received. 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.

Operating Modes

The following sections provide information on the operation modes of the interface.

General Call Addressing

The TWI controller always decodes and acknowledges a general call address if it is enabled as a slave (TWISEN) and if general call is enabled using the TWIGCE bit. General call addressing (0x00) is indicated by the setting of the GCALL bit, and by the nature of the transfer, the TWI con-

troller 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 using the `TWIFAST` bit, the following timings are modified to meet the electrical requirements.

- Serial data rise times before arbitration evaluation (t_r)
- Stop condition setup time from serial clock to serial data (t_{SUSTO})
- Bus free time between a stop and start condition (t_{BUF})

Interrupts

The following sections provide information on the TWI and interrupt generation. [Table 21-4](#) provides an overview of TWI interrupts.



If TWI interrupts are routed via the DPI interrupt, programs do not need to also read the `DPI_IRPTL` register for interrupt acknowledge. A write to clear the `TWIIRPTL` register also clears the `DPI_IRPTL` register.

Interrupts

Table 21-4. TWI Interrupt Overview

Interrupt Source	Interrupt Condition	Interrupt Completion	Interrupt Acknowledge	Default IVT
DPI TWI (TX/RX)	– Master (TX completion, TX/RX buffer service, error) – slave (initiative, completion, overflow, error)	Internal transfer completion	W1C (Write one to clear) TWI-IRPTL register + RTI instruction	P14I
TWI(TX/RX)	– Master (TX completion, TX/RX buffer service, error) – slave (initiative, completion, overflow, error)	Internal transfer completion	W1C (Write one to clear) TWI-IRPTL register + RTI instruction	Need to route TWII (PICRx) to any PxxI

Interrupt Routing

The following sections describe the various possibilities for routing a TWI interrupt to the interrupt vector table (IVT).

DPI

The TWI interrupt is combined into the digital peripheral interface (DPI) interrupt. The `DPI_IRPTL` register determines whether an interrupt is generated. Listing 21-1 shows an example of how to enable the TWI over the DPI.

Listing 21-1. Enabling DPI TWI Interrupts

```
bit set model IRPTEN;          /* enables global interrupts */  
bit set imask DPII;           /* unmasks DPI interrupt */|  
ustat1 = TWI_INT;  
dm(DPI_IRPTL_RE) = ustat1;    /* unmasks TWI interrupt */
```

TWI

The TWI receive and transmit interrupts can also be programmed through the peripheral interrupt control registers (`PICRx`) as separate inter-

rupts. (By default, these interrupts are not configured in the IRPTL register—the PICRx register has to be programmed to configure them.) This method shown in Listing 21-2 uses the PICR register with the code value of the UARTx_RXI or UARTx_TXI interrupts.

Listing 21-2. Enabling TWI Interrupts

```
bit set model IRPTEN;          /* enables global interrupts */
bit set imask P1I;            /* unmasks P1I interrupt */
ustat1=dm(PICR0);           /* route TWII 0x17 to P1I */
bit set ustat1 P1I4|P1I2|P1I1|P1I0;
bit clr ustat1 P1I3;
dm(PICR0)=ustat1;
```

Interrupt Sources

The six different types of interrupts used by the TWI are grouped according to master, slave or error operation. Those used in slave operation are:

- Transfer Initiate
- Transfer Complete

and for master operation:

- Transfer Complete
- TX/RX Buffer service

and for error operation:

- Transfer Error
- Transfer Overflow

For interrupt execution, the specific TWIRXINT receive bit or the specific TWITXINT transmit bit must be enabled in the TWIIMASK register. The

Debug Features

IMASK or LIRPTL registers must also be configured based on the programmable interrupt to be used. The ISR needs to clear the status bits of the TWIIRPTL register by explicitly writing 1 into the status bit (W1C) as shown in [Listing 21-3](#).

Listing 21-3.

```
TWI_ISR:  
ustat1 = dm(TWIIRPTL);  
bit set ustat1 TWITXINT;  
dm(TWIIRPTL) = ustat1; /* W1C to clear TWI TX interrupt */  
instruction;  
instruction;  
rti;
```

Debug Features

The following section provides information on debugging features available with the TWI.

Buffer Hang Disable

To support debugging buffer transfers, the processors have a buffer hang disable (BHD) bit in the TWIFIFOCTL register. When set (=1), this bit prevents the processor core from detecting a buffer-related stall condition, permitting debugging of this type of stall condition. For more information, see [“Buffer Hang Disable \(BHD\)” on page 10-54](#).

Loop Back Routing

The controller supports an internal loop back mode by using the SRU. For more information, see [“Loop Back Routing” on page 9-40](#).

Effect Latency

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

Write Effect Latency

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

TWI Effect Latency

After the TWI registers are configured the effect latency is 1.5 PCLK cycles minimum and 2 PCLK cycles maximum.

Programming Model

The following sections include information for general setup, slave mode, and master mode, as well as guidance for repeated start conditions.

General Setup

General setup refers to register writes that are required for both slave mode and master mode operation. General setup should be performed before either the master or slave enable bits are set.

Programs should enable the TWI controller through the `TWIMITR` register and set the prescale value. Program the prescale value to the binary representation of $f_{PCLK}/10$ MHz.

All values should be rounded up to the next whole number. The `TWIEN` enable bit must be set. Note that once the TWI controller is enabled, a bus busy condition may be detected.

Slave Mode

When enabled, slave mode supports both receive and transmit data transfers. It is not possible to enable only one data transfer direction and not acknowledge (NAK) the other. This is reflected in the following setup.

1. Program the `TWISADDR` register. The appropriate 7 bits are used in determining a match during the address phase of the transfer.
2. Program the `TXTWI18` or `TXTWI16` register. These are the initial data values to be transmitted in the event the slave is addressed as a transmitter. This is an optional step. If no data is written and the slave is addressed and a transmit is required, the `TWI_CLOCK` is stretched and an interrupt is generated.
3. Program the `TWIFIFOCTL` register. Indicate if transmit (or receive) FIFO buffer interrupts should occur with each byte transmitted (received) or with each 2 bytes transmitted (received).
4. Program the `TWIIMASK` register. Enable bits associated with the desired interrupt sources. As an example, programming the value `0x000F` results in an interrupt output to the processor when a valid address match is detected, a valid slave transfer completes, a slave transfer has an error, or a subsequent transfer has begun but the previous transfer has not been serviced.
5. Program the `TWISCTL` register. This prepares and enables slave mode operation. As an example, programming the value `0x0005` enables slave mode operation, requires 7-bit addressing, and indicates that data in the transmit FIFO buffer is intended for slave mode transmission.

[Table 21-5](#) shows what the interaction between the TWI controller and the processor might look like when the slave is addressed as a receiver.

Table 21-5. Slave Mode Setup Interaction (Slave Addressed as Receiver)

TWI Controller Master	Processor
Interrupt: TWISINIT – Slave transfer has been initiated.	Change on the next sides always. Interrupt Acknowledge: W1C the TWIIRPTL register.
Interrupt: TWIRXS – Receive buffer has 1 or 2 bytes (according to TWIRXINT).	Read receive FIFO buffer. Change on the next sides always. Interrupt Acknowledge: W1C the TWIIRPTL register
...	...
Interrupt: TWISCOMP – Slave transfer complete.	Read receive FIFO buffer. Acknowledge: Clear interrupt source bits.

Master Mode Clock Setup

Master mode operation is set up and executed on a per-transfer basis. An example of programming steps for a receive and for a transmit are given separately in following sections. The clock setup programming step listed here is common to both transfer types.

Program the `TWIDIV` register. This defines the clock high duration and clock low duration.

Master Mode Transmit

Follow these programming steps for a single master mode transmit:

1. Program the `TWIMADDR` register. This defines the address transmitted during the address phase of the transfer.
2. Program the `TXTWI8` or `TXTWI16` registers. This is the initial data transmitted. It is considered an error to complete the address phase of the transfer and not have data available in the transmit FIFO buffer.

Programming Model

3. Program the `TWIFIFOCTL` register. Indicate if transmit FIFO buffer interrupts should occur with each byte transmitted (8 bits) or with each 2 bytes transmitted (16 bits).
4. Program the `TWIIMASK` register. Enable the bits associated with the desired interrupt sources. For example, programming the value 0x0030 results in an interrupt output to the processor when the master transfer completes, or if a master transfer error has occurred.
5. Program the `TWIMCTL` register. This prepares and enables master mode operation. As an example, programming the value 0x0201 enables master mode operation, generates a 7-bit address, sets the direction to master-transmit, uses standard mode timing, and transmits 8 data bytes before generating a stop condition.

Table 21-6 shows what the interaction between the TWI controller and the processor might look like using this example.

Table 21-6. Master Mode Transmit Setup Interaction

TWI Controller Master	Processor
Interrupt: TWITXINT – Transmit buffer has 1 or 2 bytes empty (according to XMTINTLEN).	Write transmit FIFO buffer. Change on the next sides always. Interrupt Acknowledge: W1C the TWI-IRPTL register.
...	...
Interrupt: TWIMCOMP – Master transfer complete.	Change on the next sides always. Interrupt Acknowledge: W1C the TWI-IRPTL register

Master Mode Receive

Follow these programming steps for a single master mode transmit:

1. Program the `TWIMADDR` register. This defines the address transmitted during the address phase of the transfer.

2. Program the `TWIFIFOCTL` register. Indicate if receive FIFO buffer interrupts should occur with each byte received (8 bits) or with each 2 bytes received (16 bits).
3. Program the `TWIIMASK` register. Enable bits associated with the desired interrupt sources. For example, programming the value `0x0030` results in an interrupt output to the processor in the event that the master transfer completes, and the master transfer has an error.
4. Program the `TWIMCTL` register. Ultimately this prepares and enables master mode operation. As an example, programming the value `0x0201` enables master mode operation, generates a 7-bit address, sets the direction to master-receive, uses standard mode timing, and receives 8 data bytes before generating a stop condition.

[Table 21-7](#) shows what the interaction between the TWI controller and the processor might look like using this example.

Table 21-7. Master Mode Receive Setup Interaction

TWI Controller Master	Processor
Interrupt: TWIRXINT – Receive buffer has 1 or 2 bytes (according to RCVINTLEN).	Read receive FIFO buffer. Change on the next sides always. Interrupt Acknowledge: W1C the TWIIRPTL register.
...	...
Interrupt: TWIMCOMP – Master transfer complete.	Read receive FIFO buffer. Change on the next sides always. Interrupt Acknowledge: W1C the TWIIRPTL register.

Repeated Start Condition

In general, a repeated start condition is the absence of a stop condition between two transfers initiated by the same master. The two transfers can be of any direction type. Examples include a transmit followed by a receive, or a receive followed by a transmit. During a repeated start transfer, each interrupt must be serviced correctly to avoid errors. The following sections are intended to assist the programmer with service routine development.

Transmit/Receive Repeated Start Sequence

Figure 21-11 illustrates a repeated start data transmit followed by a data receive sequence.

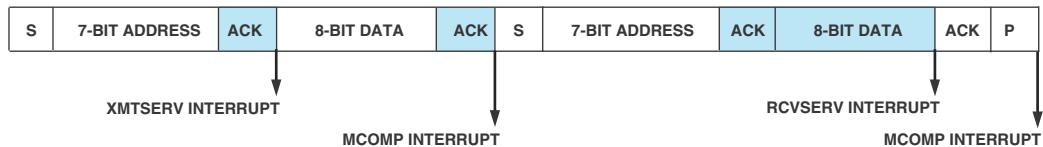


Figure 21-11. Transmit/Receive Data Repeated Start

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 21-12](#) illustrates a repeated start data receive followed by a data transmit sequence. The shading indicates the slave has the bus.

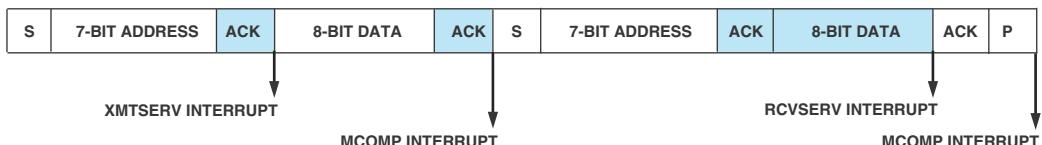


Figure 21-12. Receive/Transmit Data Repeated Start

Electrical Specifications

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.

- **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.

22 POWER MANAGEMENT

Power management is a vital tool that system designers can employ to control internal and external clocking and maximize power savings.

Features

The following list describes the power management features.

- The PLL has various multiplier and divisor settings to generate a flexible core clock.
- Allows changes to the output clock during runtime.
- RESETOUT pin can be used for boot handshake or as a debug aid.
- Resetting the PLL is possible without performing a new power-up sequence.
- Power savings controls the shut-down of individual clocks to peripherals.

Register Overview

Power Management Control Register (PMCTL). Governs the operation of the PLL and configures and controls all PLL settings.

Power Management Control Register 1 (PMCTL1). This register controls the peripheral's clocks.

Phase-Locked Loop (PLL)

The following sections describe the clocking system of the SHARC processor. This information is critical to ensure designs that work correctly and efficiently.

Functional Description

To provide the clock generation for the core and system, the processor uses an analog PLL with programmable state machine control. The PLL design serves a wide range of applications. It emphasizes embedded applications and low cost for general-purpose processors, in which performance, flexibility, and control of power dissipation are key features. This broad range of applications requires a range of frequencies for the clock generation circuitry. The input clock may be a crystal, an oscillator, or a buffered, shaped clock derived from an external system clock oscillator. The clock system is shown in [Figure 22-1](#).

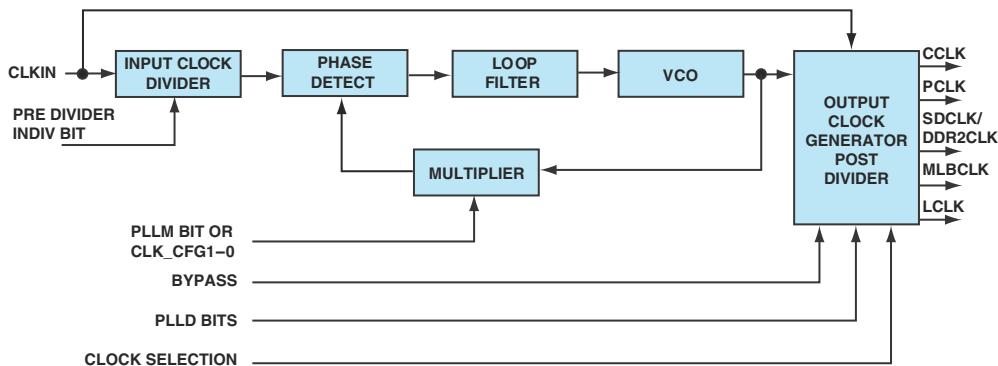


Figure 22-1. Clocking System

Subject to the maximum VCO frequency, the PLL supports a wide range of multiplier ratios of the input clock, **CLKIN**. To achieve this wide

multiplication range, the processor uses a combination of programmable multipliers in the PLL feedback circuit and output configuration blocks.

The processor uses an on-chip, phase-locked loop (PLL) to generate its internal clock, which is a multiple of the `CLKIN` frequency. The PLL requires some time to achieve phase lock and `CLKIN` must be valid for a minimum time period during reset before the `RESET` signal can be deasserted. For information on minimum clock setup, external crystal use, and range for any given `CLKIN` frequency, see the appropriate product data sheet.



A detailed diagram along with specific equations on the derivation of VCO frequency with reference to `CLKIN` can be found in the appropriate product data sheet.

PLL Input Clock

If an external clock oscillator is used, it should NOT drive the `CLKIN` pin when the processor is not powered. The clock must be driven immediately after power-up; otherwise, internal gates stay in an undefined (hot) state and can draw excess current. After power-up, allow sufficient time for the oscillator to start up, reach full amplitude, and deliver a stable `CLKIN` signal to the processor before the reset is released. This may take several milliseconds and depends on the choice of crystal, operating frequency, loop gain and capacitor ratios. For details on timing, refer to the appropriate product data sheet.

Pre-Divider Input

This unit divides the PLL input clock by 2 if enabled (using the `INDIV` bit). The pre-divider input is part of the PLL loop, therefore, if a program changes the PLL input clock (affecting the VCO frequency), the PLL must be put in bypass mode before the change is made. This is described in “[Bypass Mode](#)” on page 22-7.

PLL Multiplier

The PLL multiplier is controlled by hardware or software and based on the PLL multiplier settings below.

- Hardware—through the clock configuration pins (`CLK_CFG1-0`)
- Software—the hardware settings are overridden through the `PLL_M` bits

PLL M Hardware Control

On power-up, the `CLK_CFG1-0` pins are used to select core to `CLKIN` ratios which cannot be changed during runtime. After booting however, numerous other ratios (slowing or speeding up the clock) can be selected through software control.

For information on the internal clock to `CLKIN` frequency ratios supported by the various processors, see the product specific data sheet.

PLL M Software Control

Programs control the PLL through the `PMCTL` register. The PLL multiplier (`PLL_M`) bits can be configured to set a multiplier range of 0 to 63. This allows the PLL to be programmed dynamically in software to achieve a higher or slower core instruction rate depending on a particular system's requirements.

The reset value of the `PLL_M` bits is derived from the `CLK_CFG1-0` pin multiply ratio settings. This value can be reprogrammed in the boot kernel to take effect immediately after start-up.

PLL VCO

The VCO is the PLL output stage of the PLL. It feeds the output clock generator which provides core and peripheral clocks as shown in [Table 22-1](#). Two settings have an impact on the VCO frequency:

- The `INDIV` bit enables the `CLKIN` input pre-divider by 2.
- The `PLLM` bits and the `CLK_CFG1-0` pins control the PLL multiplier unit.

Changing the VCO frequency requires a new condition for the PLL circuitry. Therefore, the core needs to wait a specific settling time in bypass mode before it can be released for further activities (typically 4096 `CLKIN` cycles).

Table 22-1. VCO Encodings

PLLM Bit Settings	VCO Frequency ¹	
	INDIV = 0	INDIV = 1
0	128x	64x
1	2x	1x
2	4x	2x
N = 3–62	2Nx	Nx
63	126x	63x

1 For operational limits for the VCO clock see the appropriate product data sheet.

Output Clock Generator

The output clock generator post divides the VCO clock to the core ratio or peripherals ratio and synchronizes all output clocks. It is fed with the VCO clock and does not provide any feedback back to the PLL circuit.

Phase-Locked Loop (PLL)

If the `DIVEN` bit is set, new post divider ratios are picked up on the fly and the clocks smoothly transition to their new values within 14 core clock (`CCLK`) cycles.



Post divider ratio changes (`PLL0` bits) do not require bypass mode.

The output clock generator block also controls bypass mode. For a description of the `PMCTL` bits, see “[ADSP-2146x Power Management Registers](#)” on page A-6 and “[ADSP-2147x/ADSP-2148x Power Management Registers](#)” on page A-12.

Core Clock (CCLK)

The `PLL0` bits define the VCO output clock to core clock ratio to build the processor core clock (`CCLK`). The post divider can be changed any time and new division ratios are implemented on the fly.

IOP Clock (PCLK)

The peripheral clock is derived from the core clock with a fixed post divisor of 2. This clock is the master clock for all peripherals (except SDRAM) including the I/O processor (IOP).

SDRAM/DDR2 Clock (SDCLKx/DDR2_CLK)

The DDR2/ SDRAM clock is derived from the core clock. The default divisor is 2. After `RESET` is deasserted, both DDR2/SDRAM output clocks are driven with `CCLK/2 = PCLK`, independent of the DDR2/SDRAM controller configuration. Note that the `DIVEN` bit needs to be changed whenever there is a change from the default ratio.

Default PLL Hardware Settings

Table 22-2 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-locked loop. Note that the goal in selecting a particular clock ratio for an application is to provide the highest permissible internal frequency for a given CLKIN frequency. For more information on available clock rates, see the appropriate product data sheet.

Table 22-2. Selecting Core to CLKIN Ratio (ADSP-2146x)

	Typical Crystal and Clock Oscillators Inputs					
	12.500	16.667	25.000	33.333	40.000	50.000
Clock Ratios (CLK_CFG Pins)	Core CLK (MHz) ¹					
6:1 ²	N/A	100	150	200	240	300
16:1	200	266.66	400	N/A	N/A	N/A
32:1	400	N/A	N/A	N/A	N/A	N/A

1 For operational limits for the core clock frequency see the appropriate product data sheet.

2 For ADSP-2147x and ADSP-2148x models, the ratio is 8:1.

Operating Modes

The following sections provide information on the various options for clock operation.

Bypass Mode

Bypass mode must be used if any runtime VCO clock change is required. Setting the PLLBP bit bypasses the entire PLL circuitry. In bypass mode, the core runs at CLKIN speed. Once the PLL has settled into the new VCO frequency, (which may take 4096 CLKIN cycles) the PLLBP bit may be cleared to release the core from bypass mode. [For more information, see “Back to Back Bypass” on page 22-17.](#)



Only VCO frequency changes require bypass mode, therefore this mode is not intended as a standard operating mode.

Power-Up Sequence

Normal Mode

The normal mode is the regular mode and is effective if the `PLLBP` bit is cleared. In normal mode the PLL has locked and multiplies `CLKIN` to the desired VCO clock. The output clock generator post divides and provides the clock tree to the I/O.



The change of PLL frequency can happen at any time (for example after power-up or during operation).

Clocking Golden Rules

The five rules below should be followed to ensure proper processor operation.

1. After power-up the `CLK_CFG` pins should not exceed the maximum core speed.
2. Software should guarantee minimum/maximum `CCLK` speed.
3. Software should guarantee maximum VCO clock speed.
4. Bypass requires 4096 `CLKIN` cycles.
5. Post divider changes require 14 `CCLK` cycles.

Power-Up Sequence

The proper power-up sequence is critical to correct processor operation as described in the following sections.

PLL Start-Up

Before the PLL can start settling, the **RESET** signal should be asserted for several micro-seconds under the following conditions. For PLL information, see the appropriate product data sheet.

- Valid and stable core voltage (**VDDINT**)
- Valid and stable I/O voltage (**VDDEXT** and **VDD_DDR2**)
- Valid and stable clock input (**CLKIN**)

The chip reset circuit is shown in [Figure 22-2](#). The PLL needs time to lock to the **CLKIN** frequency before the core can execute or begin the boot process. A delayed core reset signal (**RESETOUT**) is triggered by a 12-bit counter after **RESET** is transitioned from low to high (approximately 400 μ s for minimum **CLKIN**). The delay circuit is activated at the same time the PLL is triggered for settling after reset is deasserted.

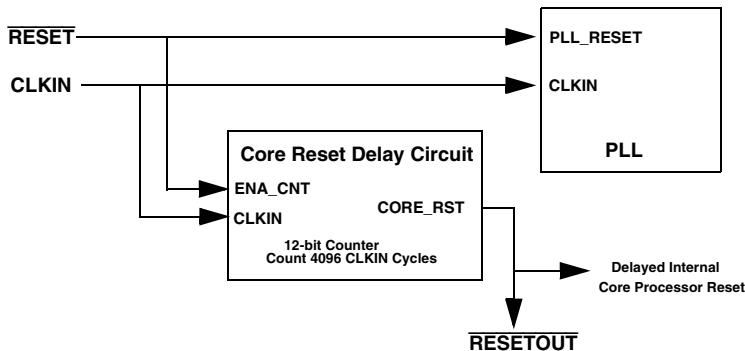


Figure 22-2. Chip Reset Circuit

After the external processor **RESET** signal is deasserted, the PLL starts settling. The rest of the chip is held in reset for 4096 **CLKIN** cycles after **RESET** is deasserted by an internal reset signal.

Power Management

 The advantage of the delayed core reset is that the PLL can be reset any number of times without having to power down the system. If there is a brownout situation, the external watchdog circuit only has to control the `RESET` signal. For more information on device power-up, see the appropriate product data sheet.

Power Management

The processor allows systems to shut down the clock to the different modules in the peripheral domain in order to save power if the peripherals are not required by the application.

Peripherals

All internal clocks to the peripherals are enabled by default. However, they can be disabled using the `PMCTL1` register.

The `PMCTL1` register allow programs to disable the clock source to a particular processor peripheral (for example the external port) to further conserve power. Programs can use the `PMCTL1` register to turn the specific peripheral off after the application no longer needs it. For a complete register description, see “[ADSP-2146x Power Management Registers](#)” on [page A-6](#) and “[ADSP-2147x/ADSP-2148x Power Management Registers](#)” on [page A-12](#).

DAI Routing Unit

To further preserve power, the clock to the DAI routing unit can be disabled using the `DAIOFF` bit in the `PMCTL1` register. If, for example, a system requires the SPORTs, the DAI system routing can be configured and then bit 4 can be cleared to disable the routing block. Note that the SPORT continues to operate as long as the SPORT clocks themselves are not shut down.

External Port Control

The `EPOFF` bit in the `PMCTL1` register allows programs to disconnect the clocks to the SDRAM and AMI modules in order to save power if the controllers are not used. Note that if the SDRAM/DDR2 controller is used but pauses, the self-refresh mode also helps to reduce power consumption. For more information, see “[SDRAM Controller \(ADSP-2147x/ADSP-2148x\)](#)” on page 3-17.

Disabling the SDRAM Controller

If the SDC is not used, the `DSDCTL` bit can be disabled to stop the clock and reduce power dissipation.



The `DSDCTL` bit must be set (=1) in products without the SDRAM interface to reduce power consumption. Set this bit as early as possible after booting the part.

Disabling the DDR2 Controller

If the DDR2 interface for ADSP-2146x processor is not used, the following bits should be configured in order to provide maximum power reduction.

- In the `DDR2CTL0` register, set (=1) the following bits: `DIS_DDR2CTL`, `DIS_DDR2CLK1` and `DIS_DDR2CKE` to disable the controller and its I/O pads.
- In the `DDR2PADCTL0` register (bits 9, 19 and 29) and `DDR2PADCTL1` register (bits 9 and 19), set (=1) all the `PWD` bits (power-down receiver pad).

Disconnect DAI/DPI Pin Buffers

If a DAI/DPI pin is not being used, its pin enable (for example `DAI_PBENxx_I`) and its input (`DAI_PBxx_I`) for its pin buffer should be connected to low.

Disable the S/PDIF Receiver

Disable the S/PDIF receiver and its digital PLL from the S/PDIF receiver. By default the S/PDIF receiver is enabled. If not required, set the `DIR_RESET` bit (=1) to disable the digital PLL which may produce unwanted switching noise if enabled.

Example for Clock Management

[Listing 22-1](#) shows a method for using the power saving features in the SHARC processors for the SPI.

Listing 22-1. Power Savings for the SPI Module

```
ustat2 = dm(PMCTL1);
bit set ustata2 SPIOFF; /* disable internal peripheral clock for
                           SPI module. /
dm(PMCTL1) = ustata2;
```

General Notes on Power Savings

The following are some additional methods for reducing power.

- The lower the operation frequency, the lower the power consumption. The core and peripherals should be operated at the lowest frequency that meets the system's requirements. Active power is proportional to the processor's core clock frequency.
- Reducing the case temperature lowers leakage power. Leakage power increases exponentially to junction temperature.
- For core pauses programs should execute the `IDLE` instruction. Note that an interrupt is required to release the processor from `IDLE`.
- Don't leave input pins floating. In some cases leakage draws current in the region of milliamps. For more information, consult the product specific data sheet. If an external resistor is problematic, change the input to an output if possible (flag input).
- If a DAI/DPI pin is not being used, its pin enable (for example `DAI_PBENxx_I`) and its input (`DAI_PBxx_I`) for its pin buffer should be connected to low.
- Disable the S/PDIF receiver and its digital PLL from the S/PDIF receiver. By default the S/PDIF receiver is enabled. If not required set the `DIR_RESET` bit (=1) to disable the digital PLL which may produce unwanted switching noise if enabled.

Programming Models

For SDRAM programming models, see “[SDRAM Controller \(ADSP-2147x/ADSP-2148x\)](#)” on page 3-17.

Post Divider

Use the following procedure and the example shown in [Listing 22-2](#) to program or reconfigure the divider.

1. Disable any peripheral (configured with $\text{PCLK}=\text{CCLK}/2$). Note that the peripherals cannot be enabled when changing VCO to core clock ratio.
2. Select the PLLD divider by setting the PLLD bits (6–7) in the PMCTL register and enable the DIVEN bit.
3. Wait 15 CCLK cycles. During this time, the new divisor ratios are picked up on the fly and the clocks smoothly transition to their new values after a maximum of 14 core clock CCLK cycles.
4. Re-enable the peripherals.

Listing 22-2. Post Divider

```
ustat2 = dm(PMCTL);
bit clr ustat2 PLLBP;           /* bypass disabled*/
bit set ustat2 DIVEN|PLLD4;    /* set and enable post divisor */
dm(PMCTL) = ustat2;
lcntr = 15, do wait until lce;
wait: nop;
```

Multiplier and Post Divider Programming Model

There are two allowable procedures to program the VCO. The first method is shown in [Listing 22-3](#).

1. Set the PLL multiplier and divisor value and enable the divisor by setting the `DIVEN` bit.
2. After one core clock cycle, place the PLL in bypass mode by setting (= 1) the `PLLBP` bit. Clear the `DIVEN` bit while placing the PLL into bypass mode.
3. Wait in bypass mode until the PLL locks (4096 `CLKIN` cycles).
4. Take the PLL out of bypass mode by clearing (= 0) the bypass bit. Clear the `DIVEN` bit while taking the PLL out of bypass mode.
5. Wait 15 core cycles before next activity.

The second method is:

1. Set the PLL multiplier and divisor values and place the PLL in bypass mode by setting the `PLLBP` bit. The second method is shown in [Listing 22-4](#).
2. Wait in the bypass mode until the PLL locks (4096 `CLKIN` cycles).
3. Take the PLL out of bypass mode by clearing the bypass bit.
4. Wait for one core clock cycle.
5. Enable the divisor by setting the `DIVEN` bit.
6. Wait 15 core cycles before next activity.

Programming Models

Listing 22-3. VCO Programming: First Method

```
ustat2 = dm(PMCTL);
bit clr ustat2 PLLM63|PLLD16;      /* Clear the old multiplier
                                         and divider values */
bit set ustat2 DIVEN | PLLD4 | PLLM16; /* set a multiplier of
                                         16 and a divider of 4 */
dm(PMCTL) = ustat2;
bit set ustat2 PLLBP;             /* Put PLL in bypass mode.*/
bit clr ustat2 DIVEN;            /* clear the DIVEN bit */

dm(PMCTL) = ustat2;              /* The DIVEN bit should be cleared
                                         while placing the PLL in bypass mode */
waiting_loop:
r0 = 4096;                      /* wait for PLL to lock at new rate
                                         (requirement for VCO change) */
lcntr = r0, do pllwait until lce;
pllwait: nop;

ustat2 = dm(PMCTL);             /* Reading the PMCTL register value
                                         returns the DIVEN bit value as zero */

bit clr ustat2 PLLBP;           /* take PLL out of Bypass, PLL is now at
                                         new CCLK */
dm(PMCTL) = ustat2;             /* The DIVEN bit should be cleared while
                                         taking the PLL out of bypass mode */

lcntr = 15, do pllwait1 until lce;
pllwait1: nop;
```

Listing 22-4. VCO Programming: Second Method

```
ustat2 = dm(PMCTL);
bit clr ustat2 PLLM63| PLLD16;     /* Clear the old multiplier
                                         and divider values */
```

```
bit set ustat2 PLLBP | PLLD4 |PLLM16; /* set a multiplier of
                                         16 and a divider of 4 */
dm(PMCTL) = ustat2;
waiting_loop:
r0 = 4096;                                /* wait for PLL to lock at new rate
                                             (requirement for VCO change) */

lcntr = r0, do pllwait until lce;
pllwait: nop;
ustat2 = dm(PMCTL);           /* Reading the PMCTL register value
                               returns the DIVEN bit value as zero.
                               The DIVEN bit should be cleared while
                               taking the PLL out of bypass mode */

bit clr ustat2 PLLBP;      /* take PLL out of Bypass,
                             PLL is now at new CCLK */
dm(PMCTL) = ustat2;
bit set ustat2 DIVEN;        /* Enable the divider */
dm(PMCTL) = ustat2;
lcntr = 15, do pllwait1 until lce;
pllwait1: nop;
```

Back to Back Bypass

Use this steps and the example shown in [Listing 22-5](#) if the application needs to re-enter the bypass mode.

1. Disable the bypass bit in the PMCTL register.
2. Wait 6 core clock cycles.
3. Enable the bypass bit.

Programming Models

Listing 22-5. Back to Back Bypass

```
ustat3 = dm(PMCTL);
bit clr ustat3 PLLBP;
dm(PMCTL) = ustat3;      /* PLLBP is cleared */
nop;nop;nop;nop;
ustat4 = dm(PMCTL);
bit set ustat4 PLLBP;
dm(PMCTL) = ustat4;      /* PLLBP is set */
```

23 SYSTEM DESIGN

This chapter discusses different processor reset methods, boot modes and pin multiplexing. In addition, information about high speed design is illustrated with some examples of supervisor circuits used in conjunction with the SHARC processor. These topics are located in the following sections.

- “Processor Reset” on page 23-3
- “Processor Booting” on page 23-7
- “Pin Multiplexing” on page 23-28
- “High Frequency Design” on page 23-33
- “System Components” on page 23-40



Before proceeding with this chapter it is recommended that you become familiar with the SHARC core architecture. This information is presented in the *SHARC Processor Programming Reference*.

Features

The following list describes the features for reset and multiplexing.

- Five reset options: hardware, software, emulation, running reset and watchdog reset.
- Different master or slave boot mechanisms.
- Hardware and software reset for processor booting.

Pin Descriptions

- Two pin multiplexing groups: core flag pins and external port pins.
- DAI/DPI units work together with multiplexing logic provides system design flexibility.

Pin Descriptions

Refer to the appropriate product data sheet for pin information, including package pinouts for the currently available package options.

Register Overview

The following registers are used for processor reset, booting, and pin multiplexing.

Software Reset Control Register (SYSCTL). Controls the software reset mechanism.

Running Reset Control Register (RUNRSTCTL). Controls the functionality of the $\overline{\text{RESETOUT}}$ pin as running reset input.

EP Control Register (EPCTL). Controls the memory chip selects for AMI on the external port memory space during boot.

EP DMA control register (DMACx). Controls the receive configuration for external boot DMA.

AMI Control Register (AMICCTL1). Controls the AMI port configuration for external port boot mode.

Link Port Control Register 0 (LCTL0). Controls the receive DMA for linkport mode during boot.

SPI Control Register (SPICTL). Controls the configuration for SPI as master or slave during SPI boot.

SPI DMA Control Register (SPIDMAC). Configures the SPI as receive DMA which generates an interrupt during boot.

SPI Slave Select Control Register (SPIFLGx). Controls the slave select configuration for SPI as master during SPI boot.

SPI Baudrate Register (SPIBAUD). Controls the `SPICLK` frequency for master mode during boot.

Processor Reset

After power-up, a `RESET` is required to place the processor into a known good state. [Table 23-1](#) shows the differences between a hardware reset (`RESET` pin deasserted) or a software reset (setting bit 0 in the `SYSCTL` register) and gives an overview of the different reset methods.

Hardware Reset

All members of the SHARC processor family support the hardware reset controlled with the `RESET` pin. The deassertion of this pin enables the PLL and asserting it resets the PLL. In the time it takes the PLL to acquire lock (set to 4096 `CLKIN` cycles), the processor, internal memory, and the peripherals are held in reset. Upon completion of the 4096 `CLKIN` cycles, the chip is brought out of reset. This is indicated on the `RESETOUT` pin for the valid boot modes. [For more information, see “Processor Booting” on page 23-7.](#)

Table 23-1. Reset Function Overview

Reset Function	Hardware Reset	Software Reset	Running Reset
<code>RESETOUT</code> Pin	Output	Output	Input
<code>RESETOUT</code> Pulse	4096 <code>CLKIN</code> cycles asserted	2 <code>PCLK</code> cycles asserted	N/A
PLL	Yes	No	No

Processor Reset

Table 23-1. Reset Function Overview (Cont'd)

Reset Function	Hardware Reset	Software Reset	Running Reset
Core	Yes	Yes	Yes
Internal Memory ¹	No	No	No
Peripherals	Yes	Yes	Yes (except SDRAM/DDR2)
Booting	Yes	Yes	No
Power Management	Yes	No	No
Emulation Unit ²	No	No	No

- 1 Internal memory array does not have reset. Only power up/down can change array contents, (or direct read/write by the core or DMA). However, if data exists in shadow FIFOs then that data is reset with any of the above resets. The logic outside the memory array is reset by all of the above three reset types, only the memory array contents remain unchanged.
- 2 There is an independent reset ($\overline{\text{TRST}}$) for the emulation interface. Enhanced Emulation (BTC) related logic is reset by the three reset types (HW Reset, SW Reset, Running Reset). Furthermore, no other part of the emulator is affected by the reset types. $\overline{\text{TRST}}$ resets the whole emulator function, including BTC.

Software Reset

In addition to the hardware reset, there is also support for a software reset, which is asserted by setting bit 0 of the `SYSCTL` register.

Running Reset

When running reset is asserted (`RESETOUT` pin acting as an input and asserted) and recognized, note the following.

- The core-PLL is NOT reset, and continues to run.
- Internal memory SRAM contents remain unaltered.

- The processor core and peripherals are reset exactly as if a Power-on (hardware) reset is asserted, except:
 - The SDRAM/DDR2 controllers continue to run and refresh as programmed.
 - The contents of external SDRAM/DDR2 are unaffected, and retain their values prior to a running reset.
 - A system boot is NOT initiated. Instead, the program counter is cleared and program execution begins from the very first location of program memory (from the reset interrupt vector table).

Running reset allows programs to:

- Execute self-modifying code that has previously overwritten existing code in internal memory.
- Activate an external watchdog in cases where there is a malfunction or exception within a peripheral.
- Perform a context reset of the processor sufficient to restore the state, (in cases where a complete boot is not required).



The `RUNRSTCTL` register is reset only on assertion of a hardware reset, software reset, emulator reset, or by writing to the appropriate bits of the `RUNRSTCTL` register via software.

For emulation reset, see the *SHARC Processor Programming Reference*, “JTAG” chapter.

System Considerations

It is important that an external $10\text{ k}\Omega$ pull-up resistor is placed on the `RESETOUT` pin if it is intended to be used as an input for initiating a running reset as shown in [Figure 23-1](#).

-  It is also extremely important to ensure that an external device, such as a micro controller, does not drive this signal during or after coming out of a power-on or hard-reset.

[Figure 23-1](#) shows the active state of the pin during and after `RESET`. The processor is actively driving this pin as an output. If the system uses an external host or micro controller to control running reset, ensure that the external device waits until the processor driver has been internally disabled (by writing to the `RUNRSTCTL` register) before actively driving this signal at `RESET`. Connect the `RESETOUT` pin to an open-drain pin on the host side, or use an external three-state buffer.

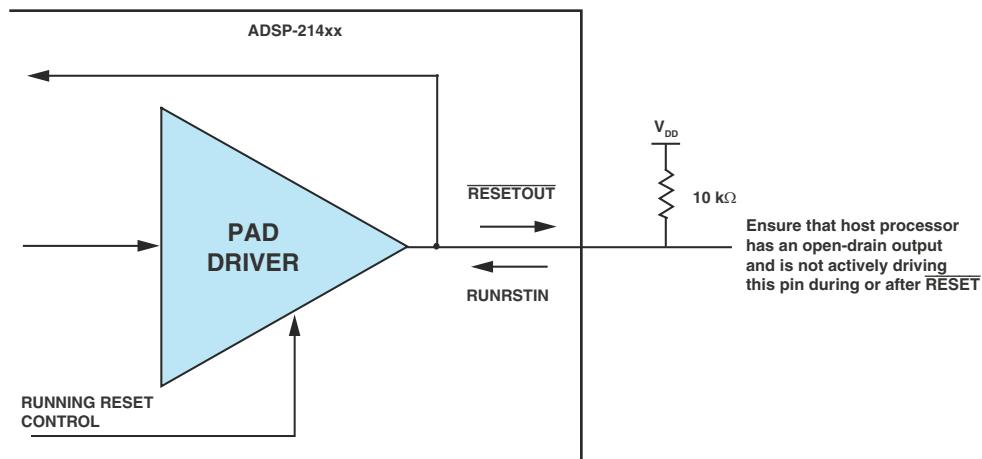
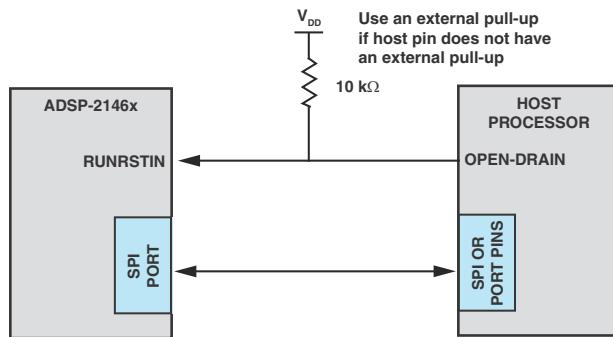


Figure 23-1. `RESETOUT` Pin Multiplexed with `RUNRSTIN`

There are several possible methods that can be used to implement running reset. The following illustrates one example of a running reset implementation involving an SHARC processor and a host processor.

External Host

In an AVR (audio-video receiver) system, a host microcontroller may communicate with the processor using the serial peripheral interface (SPI) or, if no SPI pins are available on the host device, it can use spare flag I/Os to connect with the SPI of a SHARC as shown in [Figure 23-2](#). In this case, the host implements the SPI protocol on the port pins.



[Figure 23-2](#). Example System Interface With an External Host

Processor Booting

When a processor is initially powered up, its internal SRAM is undefined. Before actual program execution can begin, the application must be loaded from an external non-volatile source such as flash memory or a host processor. This process is known as *bootstrap loading* or *booting* and is automatically performed by the processor after power-up or after a software reset.

Boot Mechanisms

In order to ensure proper device booting, the following hardware mechanisms are available on the processor.

- Peripheral boot configuration pins ($\overline{\text{BOOT_CFGx}}$) configure which peripheral boot stream is activated after power-up.
- Peripheral control and DMA parameter settings define the DMA channel which is started after $\overline{\text{RESETOUT}}$ is asserted based on the boot configuration pins.
- Peripheral interrupt is enabled after reset for the boot peripheral.
- During kernel load the core is put in IDLE. After the interrupt is generated the core jumps to reset location and starts kernel execution.

External Port Booting

The ADSP-214xx processors allow booting through the external port. The boot setting is configured through the $\overline{\text{BOOTCFG2-0}}$ pins.

The asynchronous memory interface (AMI) supports an 8-bit user boot called AMI boot. Only the $\overline{\text{MSI}}$ signal is used for AMI (FLASH/EEPROM) booting. [Table 23-2](#) shows the bit settings for AMI boot. These bits are described in detail in “[AMI Control Registers \(AMICTLx\)](#)” on page [A-21](#).

After $\overline{\text{RESETOUT}}$ deasserted, the processor starts to drive:

- ADDR23-0
- Chip select $\overline{\text{MSI}}$ to the EPROM/FLASH
- $\overline{\text{AMI_RD}}$ strobe with 23 SDCLK cycle wait states
- Read input data 7-0



The ACK pin is disabled during external port booting.

The received data streams of 4x8-bit data words are packed by the AMIRX buffer into 32-bit words least significant bit (LSB) first, and passed through the DMA's 6 deep external port buffer DFEPO into the internal memory (Figure 23-3).

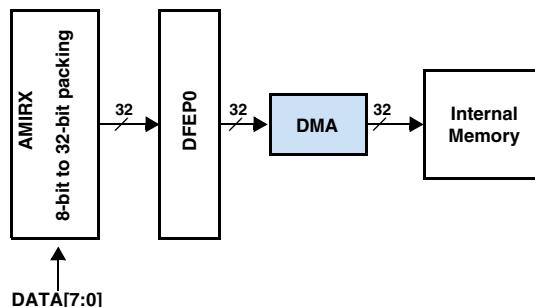


Figure 23-3. External Port Data Packing

The external port DMA channel 0 (DMAC0) is used when downloading the boot kernel information to the processor. At reset, the DMA parameter registers are initialized to the values listed in [Table 23-4](#).

In this configuration, the loader kernel is read via DMA from the FLASH. If the application needs to speed-up read accesses, programs should change the wait states (WS bits, see [Table 23-2](#)) in the kernel file. After the kernel is executed, the new wait state settings are applied and processor booting continues.

Table 23-2. AMICCTL1 Boot Settings (0x5C1)

Bit	Name	Setting
0	AMIEN	AMI enable (set = 1)
2–1	BW	Bus width = 8-bit (00)
3	PKDIS	Packing, 8-bit to 32-bit (cleared = 0)
4	MSWF	Most significant word first (cleared = 0)

Processor Booting

Table 23-2. AMICL1 Boot Settings (0x5C1) (Cont'd)

Bit	Name	Setting
5	ACKEN	ACK pin disabled (cleared = 0)
10–6	WS	23 wait state cycles = 10111
13–11	HC	Bus hold cycle at the end of write access = 000
16–14	IC	No bus idle cycle = 000
17	FLSH	Buffer holds data (cleared = 0)
20–18	RHC	Read hold cycle at the end of read access = 000
21	PREDIS	Disable Predictive Reads (cleared = 0)

Table 23-3. EPCTL Boot Settings (0xF0)

Bit	Name	Setting
0	B0SD	No SDRAM bank 0 (cleared = 0)
1	B1SD	No SDRAM bank 1 (cleared = 0)
2	B2SD	No SDRAM bank 2 (cleared = 0)
3	B3SD	No SDRAM bank 3 (cleared = 0)
5–4	EPBR	Rotating priority core vs. DMA (11)
7–6	DMAPR	Rotating priority EPDMA ch0 vs. EPDMA ch1 (11)
10–8	FRZDMA	No DMA freezing (00)
14–12	FRZCR	No core freezing (00)
18–15	DATE	No pack mode (0000) (ADSP-2147x/2148x only)
21–19	FRZSP	No SPORT DMA freezing (000)

Table 23-4. DMAC0 Boot Settings (0x1000001)

Bit	Name	Setting
0	DMAEN	DMA enabled (set = 1)
1	TRAN	Write to internal memory (cleared = 0)
2	CHEN	No DMA chaining (cleared = 0)

Table 23-4. DMAC0 Boot Settings (0x10000001) (Cont'd)

Bit	Name	Setting
3	DLEN	No delay line DMA (cleared = 0)
4	CBEN	No circular DMA (cleared = 0)
5	DFLSH	Disabled (cleared = 0)
7	WRBEN	Disabled (cleared = 0)
8	OFCEN	Disabled (cleared = 0)
9	TLEN	Disabled (cleared = 0)
12	INTIRT	Disabled (cleared = 0)
17–16	DFS	Status (cleared = 00)
20	DMAS	Status (cleared = 0)
21	CHS	Status (cleared = 0)
22	TLS	Status (cleared = 0)
23	WBS	Status (cleared = 0)
24	EXTS	External access pending (set = 1)
25	DIRS	Status (cleared = 0)

Table 23-5. Parameter Initialization for External Port Boot

Parameter Register	Initialization Value	Comment
IIEP0	IVT_START_ADDR	Start of block 0
IMEP0	0x1	
ICEP0	0x180	384 × 32-bit transfers
EIEP0	0x4000000	External memory select 1 start address
EMEP0	0x1	
ECEP0	0x180	384 × 32-bit transfers

SPI Port Booting

The SHARC processors support booting from a host processor using SPI slave mode or booting from an SPI flash, SPI PROM, or a host processor via SPI master mode. Both SPI boot modes (master and slave) support 8-, 16-, or 32-bit SPI devices. For bit settings, see the product specific processor data sheet.



- In both (master and slave) boot modes, the LSBF format is used and SPI mode 3 is selected (clock polarity and clock phase = 1). Both SPI boot modes use default routing with the DPI pin buffers. [For more information, see “DPI Default Routing” on page 9-31.](#)

Master Boot Mode

In master boot mode, the processor initiates the booting operation by:

1. Activating the SPICLK signal and asserting the SPI_FLGO_0 signal to the active low state.
2. Writing the read command 0x03 and address 0x00 to the slave device as shown in [Figure 23-5](#).

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.

SPI master booting uses the default bit settings shown in [Table 23-7](#).

Table 23-6. SPIDMAC Master/Slave Boot Settings (0x7)

Bit	Setting	Comment
SPIDEN	Set (= 1)	SPI DMA
SPIRCV	Set (= 1)	SPI receive
INTEN	Set (= 1)	SPI interrupt
SPICHEN	Cleared (= 0)	SPI DMA chaining

Table 23-6. SPIDMAC Master/Slave Boot Settings (0x7) (Cont'd)

Bit	Setting	Comment
FIFOFLSH	Cleared (= 0)	FIFO flush
INTERR	Cleared (= 0)	SPI DMA error interrupts

Table 23-7. SPICTL Master Boot Settings (0x5D06)

Bit	Setting	Comment
SPIEN	Set (= 1)	SPI enabled
SPIMS	Set (= 1)	Master device
MSBF	Cleared (= 0)	LSB first
WL	10	32-bit SPI receive shift register word length
DMISO	Cleared (= 0)	MISO enabled
SENDZ	Set (= 1)	Send zeros
SPIRCV	Set (= 1)	Receive DMA enabled
CLKPL	Set (= 1)	Active low SPI clock
CPHASE	Set (= 1)	Toggle SPICLK at the beginning of the first bit

The SPI DMA channel is used when downloading the boot kernel information to the processor. At reset, the DMA parameter registers are initialized to the values listed in [Table 23-8](#).

Table 23-8. Parameter Initialization Values for SPI Master Boot

Parameter Register	Initialization Value	Comment
SPIBAUD	0x64	SPICLK = PCLK/100
SPIFLG	0xFE01	SPI_FLG0_O used as slave-select
IISPI	IVT_START_ADDR	Start of block 0
IMSPI	0x1	32-bit data transfers
CSPI	0x180	384 × 32-bit transfers

Master Header Information

The transfer is initiated by the transferring the necessary header information on the interface (consisting of the read opcode and the starting address of the block to be transferred, which is usually all zeros). The read opcode is fixed as 0xC0 (LSBF format) and is 24-bits long. The 8-bits that are received following the read opcode should be programmed to 0xA5 (see [Figure 23-4](#)). If the 8-bits are not programmed to 0xA5 the master boot transfer is aborted. The transfer continues until 384 x 32-bit words have been transferred which may correspond to the loader program (just as in the slave boot mode).



The loader tool of VisualDSP tools automatically includes the SPI master header information (0xA5).

1. Default state of `SPICLK` signal high (out of reset).
2. Deasserting the `SPI_FLG0_0` signal (chip select) to the active low state and toggling the `SPICLK` signal.
3. Reading the read command 0x03 (MSBF format to match the LSBF format) and address 0x00 from the slave device.



Unlike previous SHARC processors, the `MOSI` pin (DPI pin 02) is three-stated for SPI master boot mode during reset.

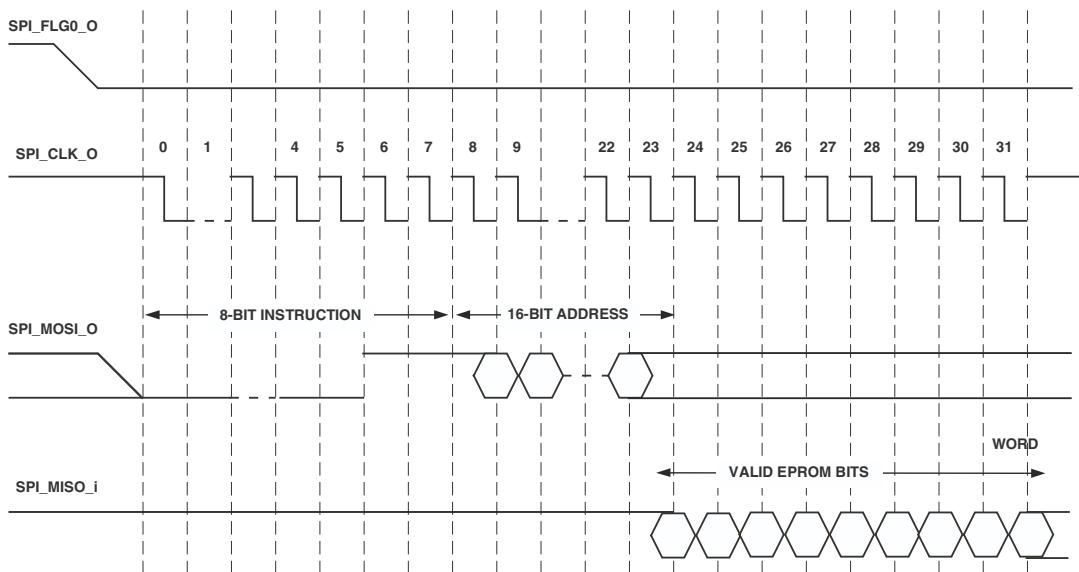


Figure 23-4. SPI Master Mode Booting Using Various Serial Devices

Slave Boot Mode

In slave boot mode, the host processor initiates the booting operation by activating the SPICLK signal and asserting the SPIIDS signal to the active low state. The 256-word kernel is loaded 32 bits at a time, through the SPI receive shift register (RXSR). To receive 256 instructions (48-bit words) properly, the SPI DMA initially loads a DMA count of 0x180 (384) 32-bit words, which is equivalent to 0x100 (256) 48-bit words.

Note that for SPI slave boot SPIIDS should only be asserted after RESETOUT has deasserted.



When in SPI slave booting mode, the SPI_DS_I input signal is controlled by the SPI host to initiate the boot transfers as shown in Table 23-9.

Processor Booting

Since the SPI host initiates the transfers, a handshake between master and slave is required for synchronization. One possible solution is to use the slave's SPI_MISO_0 signal as handshake signal. If a pause is required, the slave transmits zeros or ones to the master. Another solution is to connect this signal to the master's flag input to generate an interrupt for the same purpose.

Table 23-9. SPICTL Slave Boot Settings (0x4D22)

Bit	Setting	Comment
SPIEN	Set (= 1)	SPI enabled
SPIMS	Cleared (= 0)	Slave device
MSBF	Cleared (= 0)	LSB first
WL	10, 32-bit SPI	Receive shift register word length
DMISO	Set (= 1) MISO	MISO disabled
SENDZ	Cleared (= 0)	Send last word
SPIRCV	Set (= 1)	Receive DMA enabled
CLKPL	Set (= 1)	Active low SPI clock
CPHASE	Set (= 1)	Toggle SPICLK at the beginning of the first bit

The SPI DMA channel is used when downloading the boot kernel information to the processor. At reset, the DMA parameter registers are initialized to the values listed in [Table 23-10](#).

Table 23-10. Parameter Initialization for SPI Slave Boot

Parameter Register	Initialization Value	Comment
SPIDMAC	0x0000 0007	Enable receive, interrupt on completion
IISPI	IVT_START_ADDR	Start of block 0
IMSPI	0x1	32-bit data transfers
CSPI	0x180	384 × 32-bit transfers

SPI Boot Packing

In all SPI boot modes, the data word size in the shift register is hardwired to 32 bits. Therefore, for 8- or 16-bit devices, data words are packed into the shift register to generate 32-bit words least significant bit (LSB) first, which are then shifted into internal memory. The relationship between the 32-bit words received into the RXSPI register and the instructions that need to be placed in internal memory is shown in the following sections.

For more information about 32- and 48-bit internal memory addressing, see the “Memory” chapter in the *SHARC Processor Programming Reference*.

As shown in [Figure 23-5](#), two words shift into the 32-bit receive shift register (RXSR) before a DMA transfer to internal memory occurs for 16-bit SPI devices. For 8-bit SPI devices, four words shift into the 32-bit receive shift register before a DMA transfer to internal memory occurs.

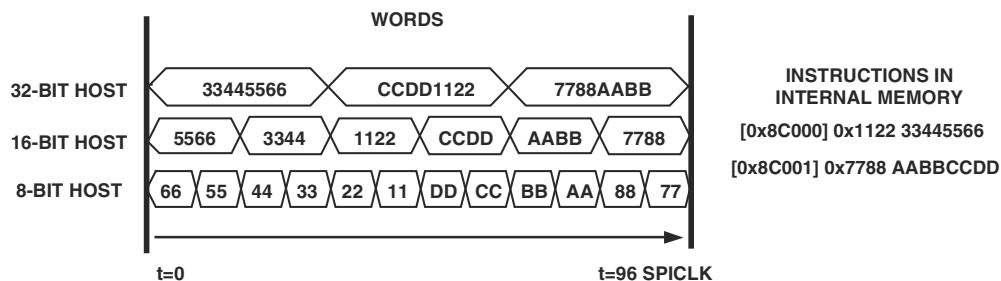


Figure 23-5. Instruction Packing for Different Hosts

When booting, the processors expect to receive words into the RXSPI register seamlessly. This means that bits are received continuously without breaks. [For more information, see “Core Transfers” on page 15-20](#). For different SPI host sizes, the processor expects to receive instructions and data packed in a least significant word (LSW) format.

Processor Booting

Figure 23-5 shows how a pair of instructions are packed for SPI booting using a 32-, 16-, and an 8-bit device. These two instructions are received as three 32-bit words.

The following sections examine how data is packed into internal memory during SPI booting for SPI devices with widths of 32, 16, or 8 bits.

32-Bit SPI Packing

Figure 23-6 shows how a 32-bit SPI host packs 48-bit instructions executed at PM addresses PMaddr0 and PMaddr1. The 32-bit word is shifted to internal program memory during the 256-word kernel load.

The following example shows a 48-bit instruction executed:

[PMaddr0] 0x112233445566
[PMaddr1] 0x7788AABBCCDD

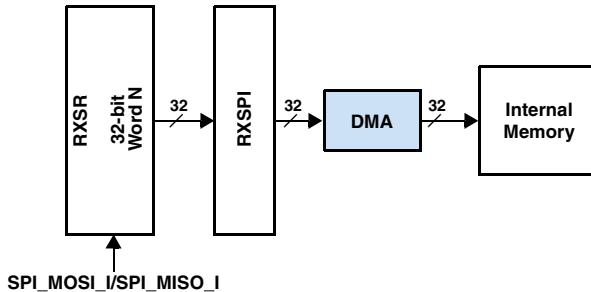


Figure 23-6. 32-Bit SPI Master/Slave Packing

The 32-bit SPI host packs or prearranges the data as:

SPI word 1= 0x33445566
SPI word 2 = 0xCCDD1122
SPI word 3 = 0x7788AABB

The initial boot of the 256-word loader kernel requires a 32-bit host to transmit 384 x 32-bit words. The SPI DMA count value of 0x180 is equal to 384 words.

16-Bit SPI Packing

Figure 23-7 shows how a 16-bit SPI host packs 48-bit instructions at PM addresses PMaddr0 and PMaddr1. For 16-bit hosts, two 16-bit words are packed into the shift register to generate a 32-bit word. The 32-bit word shifts to internal program memory during the kernel load.

The following code shows a 48-bit instruction executed:

[PMaddr0] 0x112233445566

[PMaddr1] 0x7788AABBCCDD

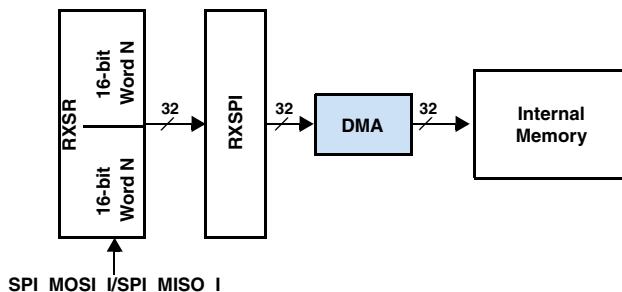


Figure 23-7. 16-Bit SPI Master/Slave Packing

The 16-bit SPI host packs or prearranges the data as:

SPI word 1 = 0x5566

SPI word 2 = 0x3344

SPI word 3 = 0x1122

SPI word 4 = 0xCCDD

SPI word 5 = 0xAABB

SPI word 6 = 0x7788

The initial boot of the 256-word loader kernel requires a 16-bit host to transmit 768 16-bit words. Two packed 16-bit words comprise the 32-bit word. The SPI DMA count value of 0x180 is equivalent to 384 words. Therefore, the total number of 16-bit words loaded is 768.

Processor Booting

8-Bit SPI Packing

Figure 23-8 shows how an 8-bit SPI host packs 48-bit instructions executed at PM addresses PMaddr0 and PMaddr1. For 8-bit hosts, four 8-bit words pack into the shift register to generate a 32-bit word. The 32-bit word shifts to internal program memory during the load of the 256-instruction word kernel.

The following code shows a 48-bit instruction executed:

```
[PMaddr0] 0x112233445566  
[PMaddr1] 0x7788AABBCCDD
```

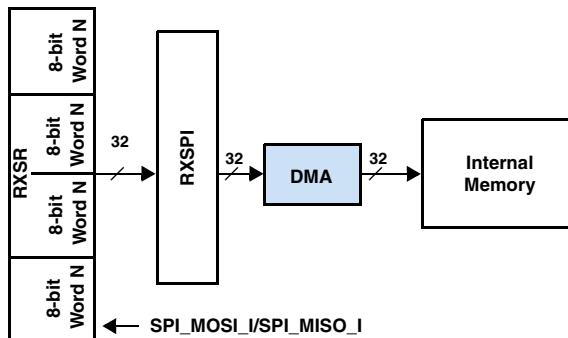


Figure 23-8. 8-Bit SPI Slave Packing

The 8-bit SPI host packs or prearranges the data as:

SPI word 1 =	0x66	SPI word 7 =	0xDD
SPI word 2 =	0x55	SPI word 8 =	0xCC
SPI word 3 =	0x44	SPI word 9 =	0xBB
SPI word 4 =	0x33	SPI word 10 =	0xAA
SPI word 5 =	0x22	SPI word 11 =	0x88
SPI word 6=	0x11	SPI word 12=	0x77

The initial boot of the 256-word loader kernel requires an 8-bit host to transmit 1536 x 8-bit words. The SPI DMA count value of 0x180 is equal

to 384 words. Since one 32-bit word is created from four packed 8-bit words, the total number of 8-bit words transmitted is 1536.

Link Port Booting

Bootling is supported through link port 0. The `BOOT_CFG2-0` values for selecting link port boot is 100.

The booting procedure is the same as any other boot mode. The acknowledge signal (`LACK0`) is asserted at `RESET` since the link port is configured as a receiver. The host initiates the transfer by toggling the link port clock (`LCLK0`). Boot data is shifted in 8-bits every clock cycle through the `LDAT0x` pins. The received data streams of 4×8 -bit is packed by the 2 deep `RXLPO` buffer into 32-bit words, least significant bit (LSB) first, and passed into the internal memory (Figure 23-9). Once the DMA is completed, a link port 0 interrupt (P1I) occurs. If `BOOT_CFG2-0` is 100 (link port 0 boot), P1I is programmed as link port 0 interrupt at reset and the interrupt is unmasked at reset. Otherwise, P1I is programmed as an `SPIHT` interrupt at reset.



For link port boot, `LCK0` should only be asserted after `RESETOUT` has deasserted.

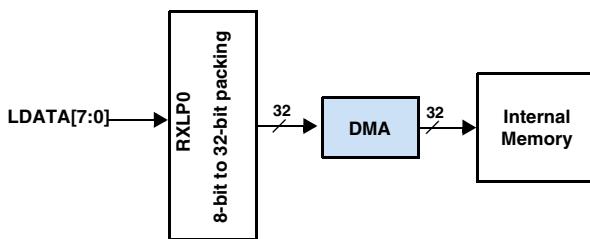


Figure 23-9. Link Port Data Packing

Processor Booting

[Table 23-11](#) shows the link port control settings after reset.

Table 23-11. LPCTL0 Boot Settings (0x403)

Bit	Name	Setting
0	LEN	Link port enabled (set = 1)
1	LDEN	DMA enabled (set = 1)
2	LCHEN	DMA Chaining (cleared = 0)
3	LTRAN	Receive operation (cleared = 0)
7	BHD	Buffer hang disabled (cleared = 0)
8	LTRQ_MSK	LP transmit request mask (cleared = 0)
9	LRRQ_MSK	LP receive request mask (cleared = 0)
10	DMACH_IRPT_MSK	LP DMA channel interrupt unmask (P1I) (set = 1)
11	LPIT_MSK	LP Invalid transmit mask (cleared = 0)
12	TXFR_DONE_MSK	External transfer done interrupt mask (cleared = 0)

The DMA parameters for the Link Port0 channel are configured as shown in [Table 23-12](#).

Table 23-12. Parameter Initialization for Link Boot

Parameter Register Elf splitter	Initialization Value	Comment
IILP0	IVT_START_ADDR	Start of block 0
IMLP0	0x1	32-bit data transfers
ICLP0	0x180	384 × 32-bit transfers

Kernel Boot Time

This section illustrates the minimum required booting time for the kernels (provided by the tools). There are five timing windows which describe together the entire boot process shown in the list below and [Table 23-13](#).

1. $\overline{\text{RESET}}$ to $\overline{\text{RESETOUT}}$ (core is in reset)
2. $\overline{\text{RESETOUT}}$ to chip select boot source (activate the boot DMA)
3. Load Kernel DMA (256 words)
4. Load application (user dependent)
5. Load IVT (256 words)

Table 23-13. Boot Times

Boot Mode	$\overline{\text{RESET}}$ to $\overline{\text{RESETOUT}}$	$\overline{\text{RESETOUT}}$ to Boot Chip Select	Kernal DMA (256 Words)	Comment
SPI Master	4096 CCLKIN	1 PCLK	$(\text{I/O} \times \text{PCLK} \div 100 + 4 \times \text{PCLK}) \times N$	$N=384, 768 \text{ or } 1536 \text{ for I/O = 32, 16 or 8}$
SPI Slave	4096 CCLKIN	Host drives signal	$(\text{I/O} \times \text{PCLK} \div 100 + 2 \times \text{PCLK}) \times N$	$N=384, 768 \text{ or } 1536 \text{ for I/O = 32, 16 or 8}$
External Port	4096 CCLKIN	5 PCLK	$24 \times \text{SDCLK} \times 1536$	
Link Port0	4096 CCLKIN	Host drives signal	$\text{LCLK0} \times 1536$	

The complete time for booting can be estimated by adding all 5 timing windows. Loading Kernel and Loading IVT both have the same size, however the default access time (wait states) for the IVT loading can be changed in the kernel by the user.

Programming Model

ROM Booting

There are two access types (modes) available for ROM booting: secured and non secured modes which are described below.

Secured ROM (hardware security switch = 1). In this mode:

- `BOOTCFG2-0` pins are ignored.
- Emulation is enabled only when the user enters a valid key.
- IIVT is placed into the internal ROM. It can be changed to the internal RAM by setting IIVT bit of `SYSCTL` register.
- Code always executes from internal ROM.

Non Secured ROM (hardware security switch = 0). In this mode:

- `BOOTCFG2-0` pins select the booting modes.
- Emulation is always enabled.
- IIVT is placed into the internal RAM except for the case where `BOOTCFG2-0 = 011`.

Programming Model

This section describes the operation of the boot process. This process is accomplished using the default loader kernel (Visual DSP Tools) to generate the boot stream. For more details, refer to the loader source files.

Running Reset

Using the SPI protocol with additional control words and commands, running reset can become an addition command from the host or from the processor as described in the following procedure.

1. The host initiates a running reset by informing the processor over the command interface.
2. The processor receives the command and completes any unfinished work which may also include writing to the RUNRSTCTL register.
3. Wait at least 5 CCLK cycles to ensure that the pin is configured as an input.
4. When the processor is ready to accept the running reset, it signals the host over the command interface.
5. The host drives the running reset input into the processor.

Running The Boot Kernel

The following sections provide information on the use of the boot kernel with the SHARC processors.

Loading the Boot Kernel Using DMA

1. At reset, the processor is hardwired (using the boot configuration pins) to load 256 x 48-bit instruction words via a DMA starting at IVT_START_ADDRESS.
2. The sequencer is put into IDLE until the boot interrupt occurs.

Programming Model

Executing the Boot Kernel

1. The DMA completes (counter zero) and the interrupt associated with the peripheral that the processor is booting from is activated.
2. The processor jumps to the applicable interrupt vector location and executes the RTI instruction located there (only).



If using your own loader kernel, you must ensure that the RTI instruction points to the IVT location of the boot peripheral.

Loading the Application

1. Once the kernel is executed (initialization of some core and external peripheral registers and such as AMI or SDRAM), the kernel prepares a DMA for further data.
2. After this the DMA starts and the core waits in IDLE until an interrupt is generated.
3. The kernel then reads the header data from a memory scratch location, decodes the header and configures a loop which loads all of the header's corresponding data.
4. Step 3 is repeated until all headers are executed.

Loading the Application's Interrupt Vector Table

1. The last header is recognized by the kernel indicating that booting has nearly finished.
2. The kernel prepares a 256 x 48-word DMA starting at IVT_START_ADDRESS.

This overrides the kernel with the application's IVT. However, the application needs to temporarily include the RTI instruction at the peripheral interrupt address, allowing a return from interrupt. Moreover, the last instruction in the final routine is a jump (db) including an IDLE.

3. The RTI instruction overrides the IVT address where user code is stored.

- i** While both DMA types (“[Loading the Boot Kernel Using DMA](#)” and “[Loading the Application’s Interrupt Vector Table](#)”) seem similar, loading the kernel is accomplished using hardware while loading the IVT is accomplished using software.
- i** It is very important to match the dedicated kernel to the dedicated boot type (for example SPI kernel and SPI boot type) in the elf-loader property page. If this is not done, the RTI instruction (in “[Loading the Application’s Interrupt Vector Table](#)”) will not be placed at the correct address. This causes execution errors.

Starting Program Execution

The processed interrupt returns the sequencer to the reset location by performing the two following steps.

1. Overriding the RTI instruction with user code.
2. Starting program execution from the reset location.

For other details relating to processor booting, see the boot loader source files that ship with the VisualDSP tools.

Memory Aliasing in Internal Memory

The boot loader takes advantage of memory aliasing which is essential to understand the boot mechanisms. For information on memory aliasing, see the *SHARC Processor Programming Reference*, “Memory” chapter.

Pin Multiplexing

During the boot process, word packing (for example 8 to 32-bit) is performed over the SPI. In other words, the kernel is not loaded directly with 256×48 -bit words, instead it is loaded with 384×32 -bit ‘packed words’ (2-column access). The same physical memory for instruction boot is loaded via DMA in normal word (NW) 2 column. However, after booting the same physical memory region is fetched by the sequencer in NW 3-column. For example the loader kernel itself has a NW 2 columns count of $256 \times 3/2 = 384$ words but the kernel is executed with 256 instruction fetches.

Note that the interrupt vector table addresses are defined as:

IVT_Start_Addr = 0x8C000 and IVT_End_Addr = 0x8C0FF.

Pin Multiplexing

The SHARC processors provide extensive functionality using a low pin count (reducing system cost). They do this through extensive use of pin multiplexing. The following sections provide information on this feature. Although the processors have the efficient and flexible DAI and DPI routing options, there are also I/O pins which are shared by some peripherals. The following sections discusses these options.



On the ADSP-2146x processors the AMI and DDR2 interfaces are completely independent (not multiplexed). Only the AMI controller address/memory selects and data pins are shared and therefore all pins discussed in this section refer to the AMI controller.

Core FLAG Pins Multiplexing

This module also includes the multiplexers of the FLAG0-3 pins shown in [Figure 23-10](#). The FLAG0-2 pins can act as core FLAG0-2 or $\overline{IRQ0-2}$, or a memory select $\overline{MS2}$ (FLAG2 pin) and the FLAG3 pin can act as a core FLAG3 or the TMREXP signal of the core timer or as a memory select $\overline{MS3}$.



Flag pins (FLG3-0) are connected as input after reset.

If more than four flags are required, they can multiplexed using the external port pins in the `SYSCTL` register or the DPI pins in the DPI registers.

For a detailed flag description refer to the *SHARC Processor Programming Reference*. [Table 23-14](#) provides information on `FLAG` function based on the settings of the memory select enable, the flag timer expired and the `FLAG2` interrupt bits in the system control register.

Table 23-14. Flag 3–2 Truth Table (SYSCTL Register)

MSEN Bit	TMREXPEN Bit	IRQ2EN Bit	FLAG3 Function	FLAG2 Function
0	0	0	FLAG3	FLAG2
0	0	1	FLAG3	$\overline{\text{IRQ2}}$
0	1	0	TMREXP	FLAG2
0	1	1	TMREXP	$\overline{\text{IRQ2}}$
1	0	0	MS3	$\overline{\text{MS2}}$

Backward Compatibility

The `FLAG/IRQ` (0, 1, 2, 3) pins retain their old functionality and programming. No changes are required for old programs. The select lines for multiplexes are controlled by the `SYSCTL` register. [For more information](#), see “[System Control Register \(SYSCTL\)](#)” on page A-4.

External Port Pin Multiplexing

Various peripherals use the external port for off-chip communication. These peripherals use multiplexed I/O pins and have the (functions) shown:

- External Port (AMI/SDRAM/DDR2)
- PDAP (input)

Pin Multiplexing

- FLAGS (I/O)
- PWM channels (output)

Multiplexed External Port Pins

The external port address and data pins are used to multiplex the external port interface with other peripherals. [Table 23-15](#) provides the pin settings.

Table 23-15. EPDATA Truth Table (SYSCTL Register)

EPDATA	ADDR31–16	ADDR15–8	DATA7–0
000		ADDR23–0	DATA7–0
001		Reserved	
010		Reserved	
011	FLAGS/PWM15–0 ¹		FLAGS15–0
100		Reserved	
101		PDAP (DATA + CTRL)	FLAGS7–0
110		Reserved	
111		Three-state all pins	

[Table 23-15](#) shows the following options.

- FLAGS can be mapped to any of the AMI pins.
- PDAP data/control can be completely moved to AMI pins (instead of DAI pins).
- PDAP, PWM signals can be mapped only to the upper bits of the AMI address pins
- FLAGS/PWM can be mapped (in groups of four) to any of the upper 31–16 AMI address pins.

Backward Compatibility

The multiplexing scheme is not backward compatible to previous SHARC processors. On previous SHARC processors only the external port data pins are multiplexed. With the ADSP-214xx processors, address and data pins of the external port are multiplexed.

Parallel Connection of Flag Pins via External Port and DPI Pins

The various external port multiplexing (shown in [Figure 23-10](#)) and DPI routing options allow situations where the flag direction paths from the core to the external port or DPI pins operates in parallel. Note that:

For FLAG3–0

- In output mode, if the same flag is mapped to both external port pins and FLAG3–0 pins, then the output is driven to both pins.
- In input mode, if the same flag is mapped to both external port pins and FLAG3–0 pins, then the input from external port pins has priority.

For FLAG15–4

- In output mode, if the same flag is mapped to both external port pins and DPI pins, then the output is driven from both pins.
- In input mode, if the same flag is mapped to both external port pins and DPI pins, then the input from the external port pins has priority.
- In input mode, if the same flags are mapped to both the upper AMI (ADDR23–8) and lower AMI (ADDR7–0, DATA7–0) pins, then the input from lower AMI pins have priority.

Pin Multiplexing

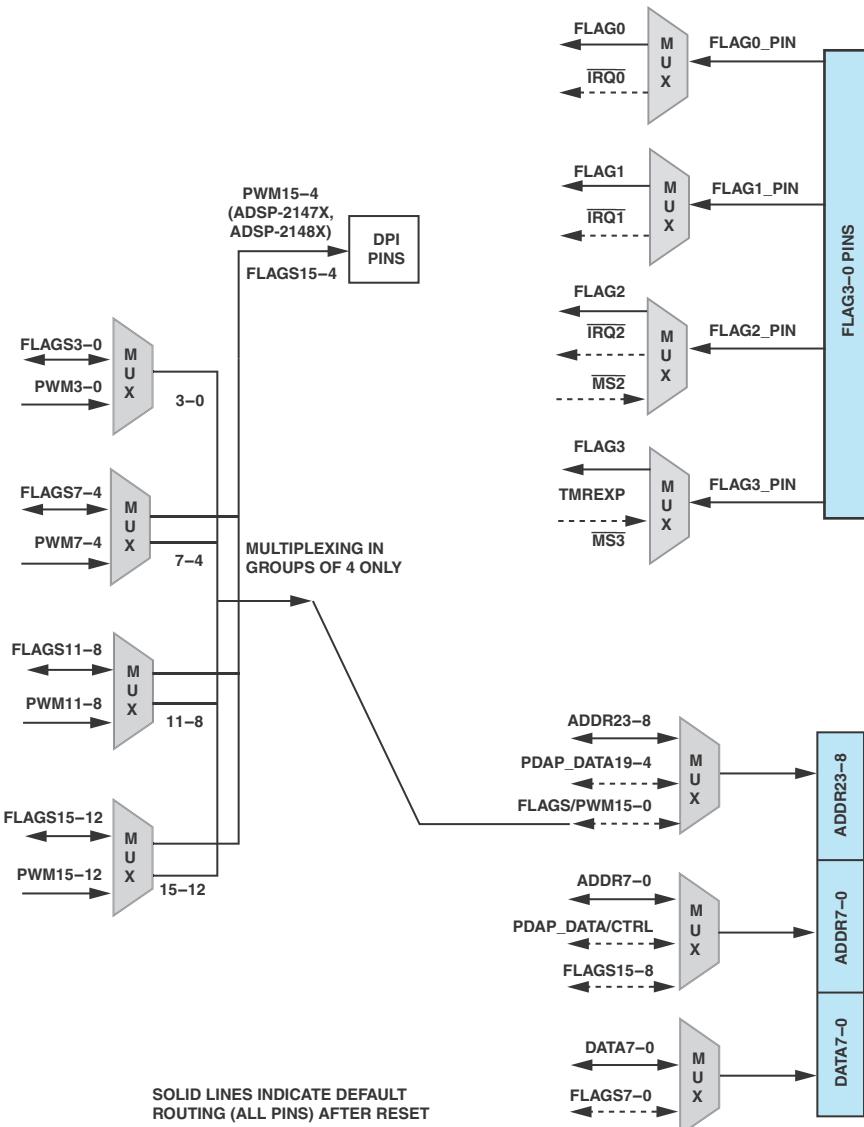


Figure 23-10. Pin Multiplexing

High Frequency Design

Because the processor must be able to operate at very high clock frequencies, signal integrity and noise problems must be considered for circuit board design and layout. The following sections discuss these topics and suggest various techniques to use when designing and debugging target systems.

Circuit Board Design

The processor is a CMOS device. It has input conditioning circuits which simplify system design by filtering or latching input signals to reduce susceptibility to glitches or reflections.

The following sections describe why these circuits are needed and their effect on input signals.

Clock Input Specifications and Jitter

The clock input signal must be free of ringing and jitter. Clock jitter can easily be introduced into a system where more than one clock frequency exists. Jitter should be kept to an absolute minimum. High frequency jitter on the clock to the processor may result in abbreviated internal cycles.

Keep the portions of the system that operate at different frequencies as physically separate as possible. The clock supplied to the processor must have a maximum rise time and must meet or exceed a high and low voltage of V_{IH} and V_{IL} , respectively.

Refer to the appropriate product data sheet for exact specifications.

RESETOUT

Circuit boards should have a test pad for the $\overline{\text{RESETOUT}}$ pin. This pin can be used as handshake signal for booting or as clock out (CLKIN frequency) for a debug aid to verify the processor is active and running.

Input Pin Hysteresis

Hysteresis (shown in [Figure 23-11](#)) is used on all SHARC input signals. Hysteresis causes the switching point of the input inverter to be slightly above 1.4 V (V_T) for a rising edge (V_{T+}) and slightly below 1.4 V for a falling edge (V_{T-}). The value of the hysteresis is approximately ± 100 mV. The hysteresis is intended to prevent multiple triggering of signals that are allowed to rise slowly, as might be expected for example on a reset line with a delay implemented by an RC input circuit. Hysteresis is not used to reduce the effect of ringing on processor input signals with fast edges, because the amount of hysteresis that can be used on a CMOS chip is too small to make a difference. The small amount of hysteresis allowed is due to restrictions on the tolerance of the V_{IL} and V_{IH} TTL input levels under worst-case conditions.

Refer to the appropriate product data sheet for exact specifications.

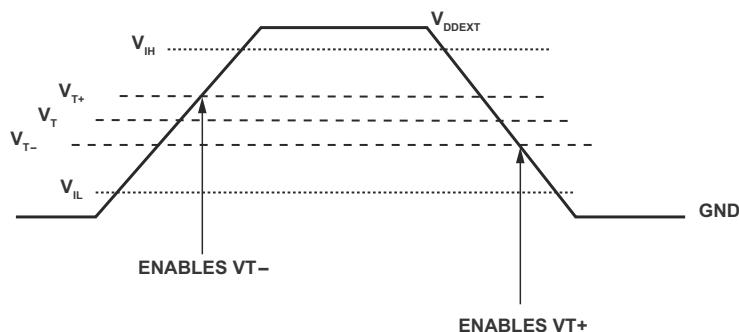


Figure 23-11. Input Pin Hysteresis

Pull-Up/Pull-Down Resistors

The pin descriptions in the product specific data sheets includes recommendations on how to handle pins on interfaces that are disabled or for unused pins on interfaces that are enabled. Generally, if internal pull-ups (PU) or pull-downs (PD) are included, the pins can be left floating. Any pin that is output only can always be left floating.

If internal pull-ups and pull-downs are not included or disabled, pins can normally still be floated with no functional issues for the device. However, this may allow additional leakage current.

Although the recommendations normally indicate using external pull-up resistors, pull-down resistors can also be used. The leakage is the same whether pull-ups or pull-downs are used. Connections directly to power or ground can be used only if the pins can be guaranteed to never be configured as outputs.

Memory Select Pins

When the multiplexed memory selects, $\overline{MS3-2}$, are enabled as outputs, the pull-up resistors are automatically enabled. For example, if $\overline{MS2}$ and $\overline{MS3}$ are used, they require that stronger external pull-up resistors are connected. For more details on resistor values, refer to the product specific data sheet.

Edge-Triggered I/O

It is recommended that GPIO output pins that are used to drive an edge-sensitive signal like an interrupt ($\overline{IRQ2-0}$, DAI/DPI pins) have series termination resistors to prevent glitches on the signal transitions. It is equally important that GPIO inputs that are edge-sensitive be driven from sources that have series termination resistors. The values for the series resistor can be determined by simulating with the IBIS models. These models can be found on the Analog Devices web site.

High Frequency Design

Asynchronous Inputs

The processor has several asynchronous inputs such as $\overline{IRQ2-0}$, FLAG3-0, ACK and the DAI/DPI pins and reset inputs \overline{RESET} , \overline{TRST} , running reset which can be asserted in arbitrary phase to the reference clocks. The processor synchronizes the reset inputs to the $CLKIN$ input while the peripheral inputs are synchronized to the $PCLK$ prior to recognizing them.

The delay associated with recognition is called the synchronization delay. Any asynchronous input must be valid prior to the recognition point in a particular cycle. If an input does not meet the setup time on a given cycle, it may be recognized in the current cycle or during the next cycle.

To ensure recognition of an asynchronous input, it must be asserted for at least one $PCLK$ cycle plus setup and hold time, except for \overline{RESET} , which must be asserted for at least four $CLKIN$ processor cycles. The minimum time prior to recognition (the setup and hold time) is specified in the appropriate product data sheet.

Decoupling and Grounding

Designs should use an absolute minimum of four bulk capacitors ($2 \times 10 \mu F$ for V_{DDINT} and $2 \times 10 \mu F$ for V_{DDEXT}). Furthermore a minimum of $20 \times 10nF$ ceramic bypass capacitors (5 per chip corner for V_{DDINT} and V_{DDEXT}).

Capacitors type, value and placement is critical—especially for floating point computations, which draw more power. If the bulk/bypass capacitors are insufficient, the power rails may drop, causing errors. Therefore sufficient capacitor backup is important.

Circuit Board Layout

This section gives recommendations to physical layouts for high speed designs.

- Place the oscillator close to the destination.
- Place the series termination close to the clock source.

For trace routing:

- Place a GND plane below the oscillator and buffer.
- Place a solid GND reference plane under the clock traces.
- Do not route the digital signals near or under the clock sources.

Other Recommendations and Suggestions

- Use more than one ground plane on the PCB to reduce crosstalk. Be sure to use lots of vias between the ground planes. One V_DD plane for each supply is sufficient. These planes should be in the center of the PCB.
- To reduce crosstalk, keep critical signals such as clocks, strobes, and bus requests on a signal layer next to a ground plane away from, or lay out these signals perpendicular to, other non-critical signals.
- If possible, position the processors on both sides of the board to reduce area and distances.

High Frequency Design

- To allow better control of impedance and delay, and to reduce crosstalk, design for lower transmission line impedances.
- Experiment with the board and isolate crosstalk and noise issues from reflection issues. This can be done by driving a signal wire from a pulse generator and studying the reflections while other components and signals are passive.

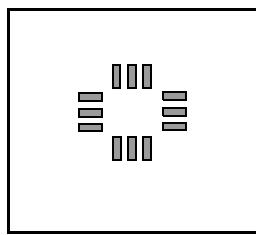
The capacitors should be placed close to the package as shown in [Figure 23-12](#). The decoupling capacitors should be tied directly to the power and ground planes with vias that touch their solder pads. Surface-mount capacitors are recommended because of their lower series inductances (ESL) and higher series resonant frequencies.

Connect the power and ground planes to the processor's power supply pins directly with vias, do not use traces. The ground planes should not be densely perforated with vias or traces as this reduces their effectiveness. In addition, there should be several large tantalum capacitors on the board.



Designs can use either bypass placement case shown in [Figure 23-12](#), or combinations of the two. Designs should try to minimize signal feedthroughs that perforate the ground plane.

BYPASS CAPACITORS ON NON-COMPONENT
(BOTTOM) SIDE OF BOARD, BENEATH DSP PACKAGE



BYPASS CAPACITORS ON COMPONENT
(TOP) SIDE OF BOARD, AROUND DSP PACKAGE

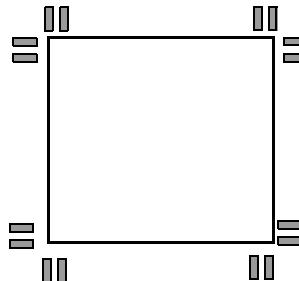


Figure 23-12. Bypass Capacitor Placement

EZ-KIT Lite Schematics

The EZ-KIT Lite® evaluation system schematics are a good starting reference. Because the EZ-KIT Lite board is for evaluation and development, extra circuitry is provided in some cases. Read the EZ-KIT Lite board schematic carefully, because sometimes a component is not populated and sometimes it has been added to make it easier to access. The design database for the SHARC processor EZ-KIT Lite boards is available online and contains all of the electronic information required for design, layout, fabrication, and assembly:

ftp://ftp.analog.com/pub/tools/Hardware/Reference_Designs

Oscilloscope Probes

When making high speed measurements, be sure to use a “bayonet” type or similarly short (< 0.5 inch) ground clip, attached to the tip of the oscilloscope probe. The probe should be a low capacitance active probe with 1 pF or less of loading. The use of a standard ground clip with four inches of ground lead causes ringing to be seen on the displayed trace and makes the signal appear to have excessive overshoot and undershoot. A 1 GHz or better sampling oscilloscope is needed to see the signals accurately.

Recommended Reading

The text *High-Speed Digital Design: A Handbook of Black Magic* is recommended for further reading. This book is a technical reference that covers the problems encountered in state-of-the-art, high frequency digital circuit design. It is also an excellent source of information and practical ideas. Topics covered in the book include:

- High-Speed Properties of Logic Gates
- Measurement Techniques

System Components

- Transmission Lines
- Ground Planes and Layer Stacking
- Terminations
- Vias
- Power Systems
- Connectors
- Ribbon Cables
- Clock Distribution
- Clock Oscillators

High-Speed Digital Design: A Handbook of Black Magic, Johnson & Graham, Prentice Hall, Inc., ISBN 0-13-395724-1.

High-Speed Signal Propagation: Advanced Black Magic, Johnson & Graham, Prentice Hall, Inc., ISBN 0-13-084408-X.

System Components

This section provides some recommendations for other components to use when designing a system for your processor.

Power Management Circuits

The ADSP-2147x and ADSP-2148x SHARC processors require a minimum of two power supplies. The ADSP-2146x processors, requires an additional supply for its DDR2 interface. The power consumption numbers are available in the respective product data sheet or EE-notes.

Refer to the following link for more information on the ADPxxxx series power supplies:

<http://www.analog.com/en/power-management/switching-regulators-integrated-fet-switches/products/index.html>

Supervisory Circuits

It is important that a processor (or programmable device) have a reliable active $\overline{\text{RESET}}$ that is released once the power supplies and internal clock circuits have stabilized. The $\overline{\text{RESET}}$ signal should not only offer a suitable delay, but it should also have a clean monotonic edge. Analog Devices has a range of microprocessor supervisory ICs with different features. Features include one or more of the following.

- Power-up reset
- Optional manual reset input
- Power low monitor
- Backup battery switching

The part number series for reset and supervisory circuits from Analog Devices are as follows.

- ADM69x
- ADM70x
- ADM80x
- ADM1232
- ADM181x
- ADM869x

System Components

A simple power-up reset circuit is shown in [Figure 23-13](#) using the ADM809-RART reset generator. The ADM809 provides an active low $\overline{\text{RESET}}$ signal whenever the supply voltage is below 2.63 V. At power-up, a 240 ms active reset delay is generated to give the power supplies and oscillators time to stabilize.

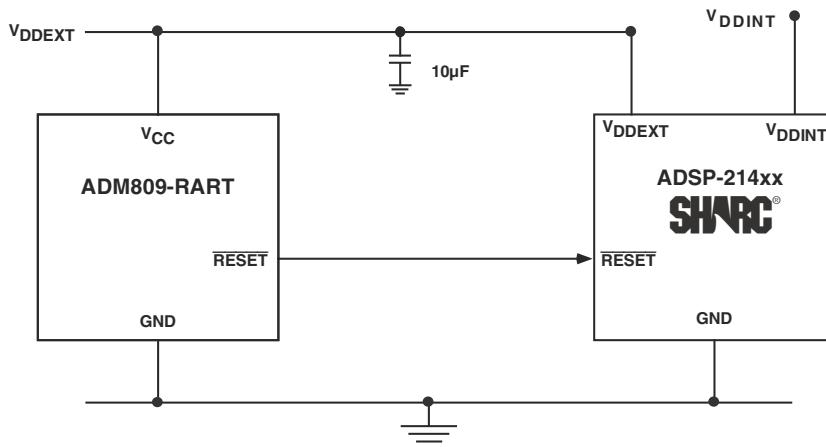


Figure 23-13. Simple Reset Generator

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 23-14](#) shows a typical application circuit using the ADM706TAR.

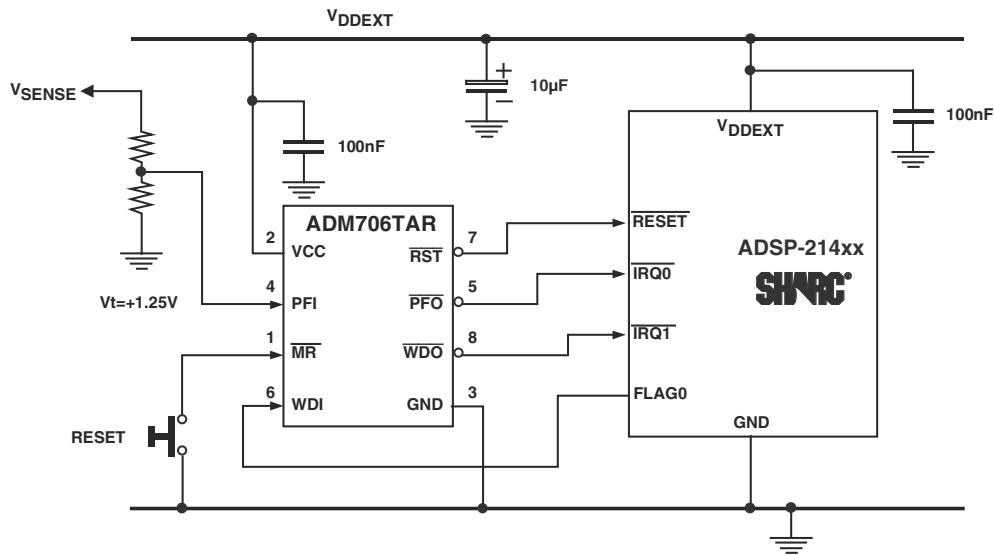


Figure 23-14. Reset Generator and Power Supply Monitor

Definition of Terms

Booting

When a processor is initially powered up, its internal SRAM and many other registers are undefined. Before actual program execution can begin, the application must be loaded from an external non-volatile source such as flash memory or a host processor. This process is known as bootstrap loading or booting and is automatically performed by the processor after power-up or after a software reset.

Boot Kernel

The boot kernel is an executable file which schedules the entire boot process. The temporary location of the kernel resides in the processor's Interrupt vector location (IVT). The IVT typically has a maximum size of 256×48 words. After booting, the kernel overwrites this area.

Definition of Terms

These kernel files (DXE, ASM) are supplied with the VisualDSP++ development tools for all boot modes. For more information on the kernels, refer to the tools documentation

Boot Master/Slave

How a processor boots is dependent on the peripheral used. See “[Processor Booting](#)” on page 23-7.

Boot Modes

The boot mode is identified by the `BOOT_CFG2-0` pins that are used in the boot process.

No Boot Mode

In legacy mode, the processor does not boot. Instead, it starts fetching instructions directly from external memory. The SHARC ADSP-214xx processors onwards do not support this mode.

ROM Boot Mode

For `BOOT_CFG2-0` pins = 011, the processor executes from internal ROM. Only specific versions of the processors support this mode.

A REGISTERS REFERENCE

The SHARC processors have general-purpose and dedicated registers in each of their functional blocks. The register reference information for each functional block includes bit definitions, initialization values, and memory-mapped addresses (for I/O processor registers). Note that this appendix only contains information for the control and or status registers. All other IOP registers (for example address, modify, count) are described in [Chapter 2, I/O Processor](#).

- “Overview” on page A-2
- “System and Power Management Registers” on page A-4
- “ADSP-2146x External Port Registers” on page A-18
- “DAI Signal Routing Unit Registers” on page A-118
- “Peripherals Routed Through the DAI” on page A-150
- “DPI Signal Routing Unit Registers” on page A-218
- “Peripherals Routed Through the DPI” on page A-231
- “Register Listing” on page A-273

When writing programs, it is often necessary to set, clear, or test bits in the processor’s registers. While these bit operations can all be done by referring to the bit’s location within a register or (for some operations) the register’s address with a hexadecimal number, it is much easier to use symbols that correspond to the bit’s or register’s name. For convenience and consistency, Analog Devices supplies a header file that provides these bit

Overview

and registers definitions. An #include file is provided with VisualDSP++ tools and can be found in the `VisualDSP/214xx/include` directory.

Overview

The I/O processor's registers are accessible as part of the processor's memory map. [“Register Listing” on page A-273](#) lists the I/O processor's memory-mapped registers and provides a brief description of each register.

Since the I/O processor registers are memory-mapped, the processor's architecture does not allow programs to directly transfer data between these registers and other memory locations, except as part of a DMA operation. To read or write I/O processor registers, programs must use the processor core registers.

The register names for I/O processor registers are not part of the processor's assembly syntax. To ease access to these registers, programs should use the header file containing the registers' symbolic names and addresses.

Register Diagram Conventions

The register drawings in this appendix provide “at-a-glance” information about specific registers. They are designed to give experienced users basic information about a register and its bit settings. When using these registers, the following should be noted.

- In cases where there are multiple registers that have the same bits (such as serial ports), one register drawing is shown and the names and addresses of the other registers are simply listed. Also, depending on peripheral (such as ASRC), if two different ASRC ports are programmed in the same register, one peripheral is defined with a `x` the other with a `y` index.

- The bit descriptions in the figures are intentionally brief, containing only the bit mnemonic, location, and function. More detailed information can be found in the tables that follow the register drawings and in the chapters that describe the particular module.
- Shaded bits are reserved.
- The VisualDSP++ tools suite contains the complete listing of registers in a header file, `def214xx.h`.
- [“Register Listing” on page A-273](#) provides a complete list of user accessible registers, their addresses, and their state at reset.

Bit Types and Settings

There are several bit types used in SHARC registers. These are described in [Table A-1](#). In general, control register bits are read-write (RW) and status register bits are read-only (RO). In exceptional cases, bit types are shown in the “Bit” column in parenthesis where for example a RO bit is used in a control register or for write-one-to-clear (W1C) bits.

Also note that the setting after reset (default setting) of most bits is 0 (cleared). In cases where this is not true, this is shown in the “Description” column in parenthesis.

Table A-1. Bit Type Usage

Bit Type	Description	Usage
RW	Read-Write	RW bits are used primarily in control registers and DMA parameter and count registers.
RO	Read-Only	RO bits are used primarily in status registers and Shadow registers/buffers for debug aid.
ROC	Read-Only-to-Clear	ROC bits are used to clear receive buffer status bits. These bits are also used on the DAI/DPI interrupt controllers. These bits are sticky and their status is only cleared after a read.

System and Power Management Registers

Table A-1. Bit Type Usage (Cont'd)

Bit Type	Description	Usage
WO	Write-Only	WO bits are used primarily in control/status register to trigger events like self-refresh or power-up sequence for SDRAM. Note that these bit type always read zero
WOC	Write-Only-to-Clear	WOC bits are used primarily in control/status register to flush data FIFOs and to clear its status bits.
W1C	Write-1-to-Clear	W1C bits are sticky bits used primarily in status registers during interrupt acknowledge for PWM and timers. These bits are sticky and their status is only cleared after a write.



Many registers have reserved bits. When writing to a register, programs may only clear (write zero to) the register's reserved bits.

System and Power Management Registers

The registers described in the following sections are used to control system wide operations and power management.

System Control Register (SYSCTL)

The SYSCTL register configures memory use, interrupts, and many aspects of pin multiplexing. (For more information, see “Pin Multiplexing” on page 23-28.) Bit descriptions for this register are shown in Figure A-1 and described in Table A-2.

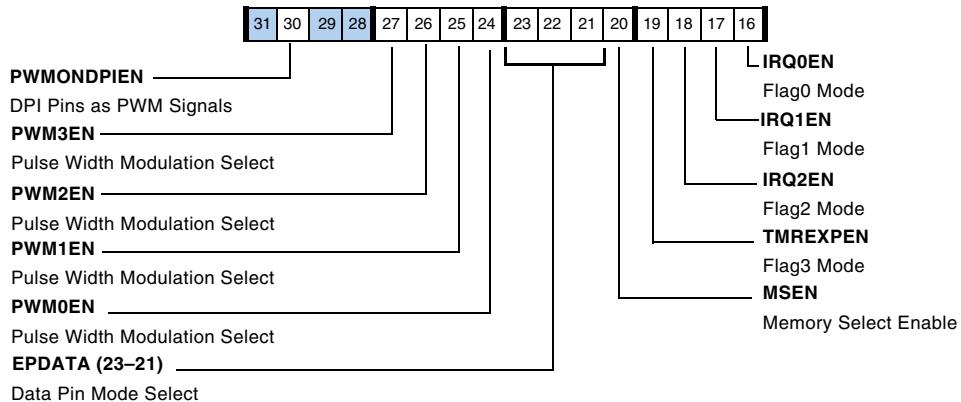


Figure A-1. SYSCTL Register

Table A-2. SYSCTL Register Bit Descriptions

Bit	Name	Description
15–0		The bits are used for controlling core function. Refer to the <i>SHARC Processor Programming Reference</i> .
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 $\overline{IRQ0}$.
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 $\overline{IRQ1}$.
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 $\overline{IRQ2}$.
19	TMREXPEN	Flag Timer Expired Mode. 0 = Flag3 pin is a general-purpose I/O pin. Permits core writes. 1 = Flag3 pin output is timer expired signal (TMREXP).
20	MSEN	Memory Select Enable. Selects between FLGx/AM _i _MSX/IRQx or TMREXP. Together with bits 19–18 generate a truth table. Detailed modes of programming for these bits are given in “ Core FLAG Pins Multiplexing ” on page 23–28. 0 = FLAG/ \overline{IRQX} pins are selected 1 = Enables FLAG2 and 3 ($\overline{IRQ2}$ and TIMEXP) as MS ₂ and 3

System and Power Management Registers

Table A-2. SYSCTL Register Bit Descriptions (Cont'd)

Bit	Name	Description
23–21	EPDATA	AMI Mode Select. Selects between multiplexed AMI, Flags, PWM and PDAP interfaces on the AMI bus. For detailed programming modes for these bits, see “ Multiplexed External Port Pins ” on page 23-30.
24	PWM0EN	Pulse Width Modulation Select. When set (=1), enables PWM3–0. For more information, see “Pin Multiplexing” on page 23-28. Reserved for ADSP-2147x and ADSP-2148x.
25	PWM1EN	Pulse Width Modulation Select. When set (=1), enables PWM7–4. For more information, see “Pin Multiplexing” on page 23-28.
26	PWM2EN	Pulse Width Modulation Select. When set (=1), enables PWM11–8. For more information, see “Pin Multiplexing” on page 23-28.
27	PWM3EN	Pulse Width Modulation Select. When set (=1), enables PWM15–12. For more information, see “Pin Multiplexing” on page 23-28.
29–28	Reserved	
30	PWMOND-PIEN	Enable PWM Signals on the DPI Pins. Enables the PWM signals on DPI pins. When this bit is set (=1), the flags (4–15) which are routed to the DPI pins can be used as PWM signals. Applicable only for ADSP-2148x and ADSP-2147x processors.
31	Reserved	

ADSP-2146x Power Management Registers

The registers described in the following sections are specific to the ADSP-2146x processors.

Power Management Control Registers (PMCTL)

The following sections describe the registers associated with the processors power management functions.

The power management control register, shown in [Figure A-2](#), is a 32-bit memory-mapped register. This register contains bits to control phase lock loop (PLL) multiplier and divider (both input and output) values, PLL bypass mode, and clock enabling control for peripherals (see [Table A-3 on page A-8](#)). This register also contains status bits, which keep track of the status of the CLK_CFG pins (read-only). The reset value of PMCTL is dependent on the CLK_CFG pins (bits 5–0 and 17–16).

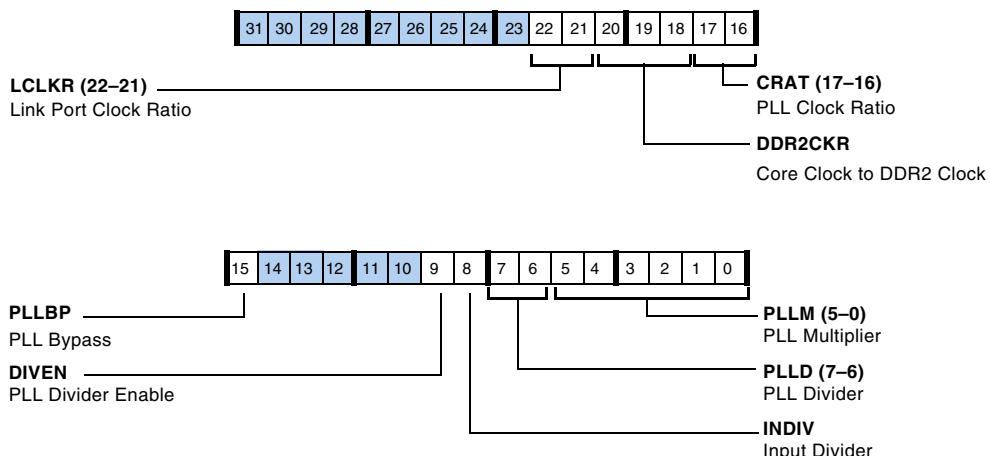


Figure A-2. PMCTL Register

System and Power Management Registers

Table A-3. PMCTL Register Bit Descriptions (RW)

Bit	Name	Description
5–0	PLLM	PLL Multiplier. PLLM = 0 PLL multiplier = 128 $0 < \text{PLLM} < 63$ PLL multiplier = $2 \times \text{PLLM}$ Reset value = CLK_CFG1–0 00 = 000110 = 6x 01 = 100000 = 32x 10 = 010000 = 16x 11 = 000110 = 6x (Reserved)
7–6	PLLD	PLL Divider (Output Post Divider). 00 = clock divider = 2 01 = clock divider = 4 10 = clock divider = 8 11 = clock divider = 16
8	INDIV	PLL Input Clock Pre Divider. 0 = Divide by 1 1 = Divide by 2
9 (WO)	DIVEN	Output Clock Divider Enable. Enables the post divider settings. 0 = Do not load PLLD 1 = Load PLLD When the PLL is programmed using the multipliers and the post dividers, the DIVEN and PLLBP bits should NOT be programmed in the same core clock cycle.
11–10	Reserved	
12	CLKOUTEN	Clockout Enable. Mux select for CLKOUT and RESETOUT. 0 = Mux output = <u>RESETOUT</u> 1 = Mux output = CLKOUT The CLKOUT functionality is not characterized and only used for test purposes.
14–13	Reserved	
15	PLLBP	PLL Bypass Mode Indication. 0 = PLL is in normal mode 1 = Put PLL in bypass mode

Table A-3. PMCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
17–16 (RO)	CRAT	PLL Configuration Ratio, CLK_CFG1-0 pins. After reset, both CLK_CFG pins define the CLKIN to core clock ratio. This ratio can be changed with the PLLM and PLLD bits. CRAT =CLK_CFG[1:0] 0 = CLK_CFG[1:0] = 00 (6:1 ratio) 1 = CLK_CFG[1:0] = 01 (32:1 ratio) 2 = CLK_CFG[1:0] = 10 (16:1 ratio) 3 = reserved
20–18	DDR2CKR	DDR2CLK Ratio. Core clock to DDR2 clock. Note that the typical minimum clock speed to typically 125 MHz. 000 = RATIO = 2 010 = RATIO = 3 100, 101, 110, 111 = reserved
22–21	LCLKR	Link Port Clock Ratio. 00 = RATIO = 2.0 01 = RATIO = 2.5 10 = RATIO = 3.0 11 = RATIO = 4.0 Reset value = 10
32–23	Reserved	

Power Management Control Register 1 (PMCTL1)

This register, shown in [Figure A-3](#) and described in [Table A-4](#), contains the bits for shutting down the clocks to various peripherals and selecting one of the three FIR/IIR/FFT accelerators.



Write to this register has effect latency of two PCLK cycles.

System and Power Management Registers

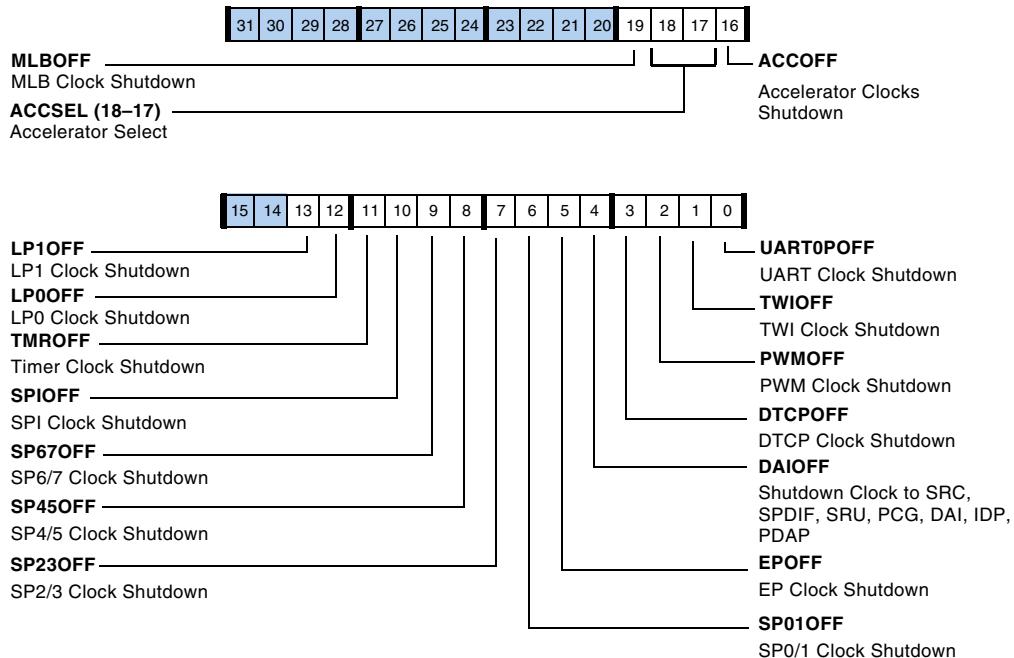


Figure A-3. PMCTL1 Register

Table A-4. PMCTL1 Register Bit Descriptions (RW)

Bit	Name	Description
0	UART0OFF	Shutdown Clock to UART. 0 = UART is in normal mode 1 = Shutdown clock to UART
1	TWIOFF	Shutdown Clock to TWI. 0 = TWI is in normal mode 1 = Shutdown clock to TWI
2	PWMOFF	Shutdown Clock to PWM. 0 = PWM is in normal mode 1 = Shutdown clock to PWM
3	DTCPOFF	Shutdown Clock to MTM/DTCP. 0 = MTM is in normal mode 1 = Shutdown clock to MTM

Table A-4. PMCTL1 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
4	DAIOFF	Shutdown Clock to DAI. Shutdown clock to DAI related peripherals—SRU, SRC, S/PDIF, PCG, IDP, PDAP 0 = DAI is in normal mode 1 = Shutdown clock to DAI
5	EPOFF	Shutdown Clock to External Port. 0 = External port is in normal mode 1 = Shutdown clock to external port
6	SP01OFF	Shutdown Clock to SPORT 0, 1. 0 = SPORTs in normal mode 1 = Shutdown clock to SPORTs
7	SP23OFF	Shutdown Clock to SPORT 2, 3. 0 = SPORTs in normal mode 1 = Shutdown clock to SPORTs
8	SP45OFF	Shutdown Clock to SPORT 4, 5. 0 = SPORTs in normal mode 1 = Shutdown clock to SPORTs
9	SP67OFF	Shutdown Clock to SPORT 6, 7. 0 = SPORTs in normal mode 1 = Shutdown clock to SPORTs
10	SP1OFF	Shutdown Clock to SPI/SPIB. 0 = SPI in normal mode 1 = Shutdown clock to SPI
11	TMROFF	Shutdown Clock to Peripheral Timer. 0 = Timer is in normal mode 1 = Shutdown clock to timer
12	LP0OFF	Shutdown Clock to Link Port 0. 0 = LP0 is in normal mode 1 = Shutdown clock to LP0
13	LP1OFF	Shutdown Clock to Link Port 1. 0 = LP1 is in normal mode 1 = Shutdown clock to LP1
15–14	Reserved	

System and Power Management Registers

Table A-4. PMCTL1 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
16	ACCOFF	Shutdown Clock to Accelerator. 0 = Accelerator is in normal mode 1 = Shutdown clock to accelerator
18–17	ACCSEL	Accelerator Select. 00 = Select FIR 01 = Select IIR 10 = Select FFT 11 = Reserved
19	MLBOFF	Shutdown Clock to Media Local Bus. 0 = MLB is in normal mode 1 = Shutdown clock to MLB
31–20	Reserved	

ADSP-2147x/ADSP-2148x Power Management Registers

The registers described in the following sections are specific to the ADSP-2147x and ADSP-2148x processors.

Power Management Control Registers (PMCTL)

The following sections describe the registers associated with the processors power management functions.

The power management control register, shown in [Figure A-4](#), is a 32-bit memory-mapped register. This register contains bits to control phase lock loop (PLL) multiplier and divider (both input and output) values, PLL bypass mode, and clock enabling control for peripherals (see [Table A-5](#)). This register also contains status bits, which keep track of the status of the CLK_CFG pins (read-only). The reset value of PMCTL is dependent on the CLK_CFG pins (bits 5–0 and 17–16).

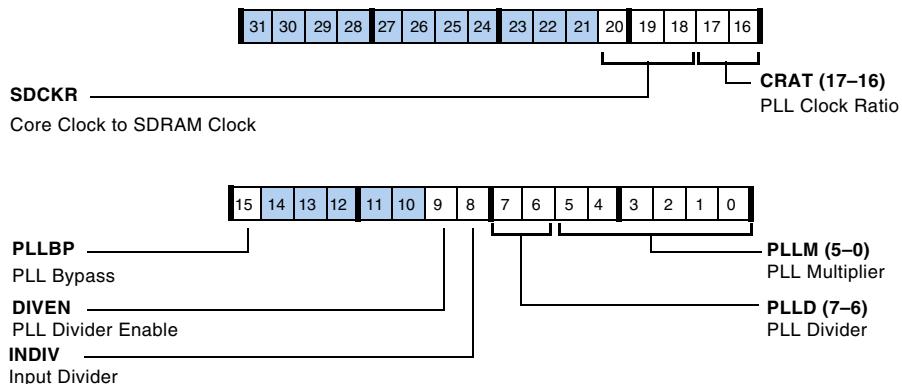


Figure A-4. PMCTL Register

Table A-5. PMCTL Register Bit Descriptions (RW)

Bit	Name	Description
5–0	PLLM	PLL Multiplier. PLLM = 0 PLL multiplier = 128 0<PLLM<63 PLL multiplier = $2 \times$ PLLM Reset value = CLK_CFG1–0 00 = 001000 = 8x 01 = 100000 = 32x 10 = 010000 = 16x 11 = 001000 = 8x
7–6	PLLD	PLL Divider (Output Clock Post Divider). 00 = clock divider = 2 01 = clock divider = 4 10 = clock divider = 8 11 = clock divider = 16
8	INDIV	PLL Input Clock Pre Divider. 0 = Divide by 1 1 = Divide by 2

System and Power Management Registers

Table A-5. PMCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
9 (WO)	DIVEN	Output Clock Divider Enable. This bit enables the post divider settings. 0 = Do not load PLLD 1 = Load PLLD When the PLL is programmed using the multipliers and the post dividers, the DIVEN and PLLBP bits should NOT be programmed in the same core clock cycle.
11–10	Reserved	
12	CLKOUTEN	Clockout Enable. Mux select for CLKOUT and $\overline{\text{RESETOUT}}$ 0 = Mux output = $\overline{\text{RESETOUT}}$ 1 = Mux output = CLKOUT Reset value = 0. The CLKOUT functionality is not characterized and only used for test purposes.
14–13	Reserved	
15	PLLBP	PLL Configuration Ratio, CLK_CFG1-0 pins. After reset, both CLK_CFG[1:0] pins defines the CLKIN to core clock ratio. This ratio can be changed with the PLLM and PLLD bits. CRAT = CLK_CFG[1:0] 0 = CLK_CFG[1:0] = 00 (8:1 ratio) 1 = CLK_CFG[1:0] = 01 (32:1 ratio) 2 = CLK_CFG[1:0] = 10 (16:1 ratio) 3 = reserved
17–16	CRAT	PLL Clock Ratio, CLKIN to CCLK. For more detail, see the PLLM and PLLDx bit descriptions in this table. Reset value = CLK_CFG1–0
20–18	SDCKR	SDRAM Clock Ratio. Core clock to SDRAM clock. 001 = SDCKR2_5 (Ratio = 2.5:1) 010 = SDCKR3 (Ratio = 3.0:1) 011 = SDCKR3_5 (Ratio = 3.5:1) 100 = SDCKR4 (Ratio = 4.0:1) 101, 110, 111 = Reserved
32–23	Reserved	

Power Management Control Register 1 (PMCTL1)

This register contains the bits for shutting down the clocks to various peripherals and selecting one of the three FIR/IIR/FFT accelerators. The core can write to bits 19–0 of this register.



Writes to this register have an effect latency of two PCLK cycles.

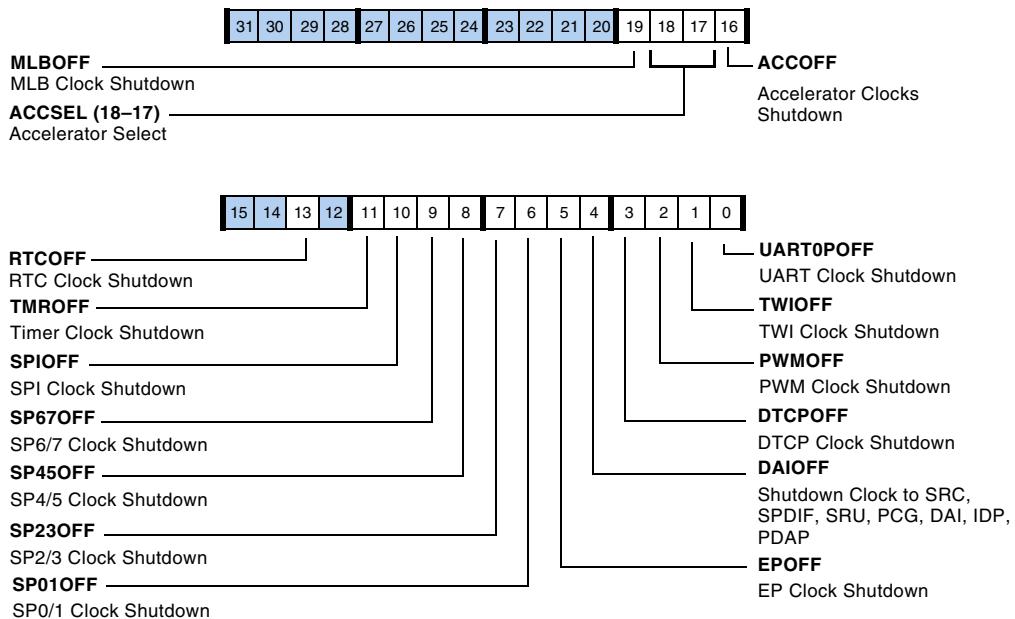


Figure A-5. PMCTL1 Register

System and Power Management Registers

Table A-6. PMCTL1 Register Bit Descriptions (RW)

Bit	Name	Description
0	UART0OFF	Shutdown Clock to UART. 0 = UART is in normal mode 1 = Shutdown clock to UART
1	TWI0OFF	Shutdown Clock to TWI. 0 = TWI is in normal mode 1 = Shutdown clock to TWI
2	PWM0OFF	Shutdown Clock to PWM. 0 = PWM is in normal mode 1 = Shutdown clock to PWM
3	DTCPOFF	Shutdown Clock to MTM/DTCP. 0 = MTM is in normal mode 1 = Shutdown clock to MTM
4	DAIOFF	Shutdown Clock to DAI. Shutdown clock to DAI related peripherals—SRU, SRC, S/PDIF, PCG, IDP, PDAP 0 = DAI is in normal mode 1 = Shutdown clock to DAI
5	EPOFF	Shutdown Clock to External Port. 0 = External port is in normal mode 1 = Shutdown clock to external port
6	SP01OFF	Shutdown Clock to SPORT 0, 1. 0 = SPORTs in normal mode 1 = Shutdown clock to SPORTs
7	SP23OFF	Shutdown Clock to SPORT 2, 3. 0 = SPORTs in normal mode 1 = Shutdown clock to SPORTs
8	SP45OFF	Shutdown Clock to SPORT 4, 5. 0 = SPORTs in normal mode 1 = Shutdown clock to SPORTs
9	SP67OFF	Shutdown Clock to SPORT 6, 7. 0 = SPORTs in normal mode 1 = Shutdown clock to SPORTs
10	SP1OFF	Shutdown Clock to SPI/SPIB. 0 = SPI in normal mode 1 = Shutdown clock to SPI

Table A-6. PMCTL1 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
11	TMROFF	Shutdown Clock to Peripheral Timer. 0 = Timer is in normal mode 1 = Shutdown clock to timer
12	Reserved	
13	RTCOFF	Shutdown Clock to Real-Time Clock. 0 = RTC is in normal mode 1 = Shutdown clock to RTC
15–14	Reserved	
16	ACCOFF	Shutdown Clock to Accelerator. 0 = Accelerator is in normal mode 1 = Shutdown clock to accelerator
18–17	ACCSEL	Accelerator Select. 00 = Select FIR 01 = Select IIR 10 = Select FFT 11 = Reserved
19	MLBOFF	Shutdown Clock to Media Local Bus. 0 = MLB is in normal mode 1 = Shutdown clock to MLB
31–20	Reserved	

Running Reset Control Register (RUNRSTCTL)

The RUNRSTCTL register is used to control the running reset functionality and is described in [Table A-7](#).

Table A-7. Running Reset Control Register Bit Descriptions (RW)

Bit	Name	Description
0	PM_RUNRST_PINEN	Configures the RESETOUT pin for RUNRST input. 0 = <u>RESETOUT</u> pin is <u>RESETOUT</u> 1 = <u>RESETOUT</u> pin is RUNRST input
1	PM_RUNRST_EN	Enable the Running Reset Functionality. If this bit is cleared, attempting to cause a running reset by toggling the <u>RUNRSTIN</u> pin has no affect 0 = Running reset disabled 1 = Running reset enabled
31–2	Reserved	

ADSP-2146x External Port Registers

The registers in the following sections are specific to the ADSP-2146x external port and include the external port, the DDR2 controller, and AMI registers.

External Port Control Register (EPCTL)

The following registers are used to control asynchronous memory interface (AMI), the DDR2 and SDRAM controllers, and the shared memory interface. 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-6](#) and described in [Table A-8](#).

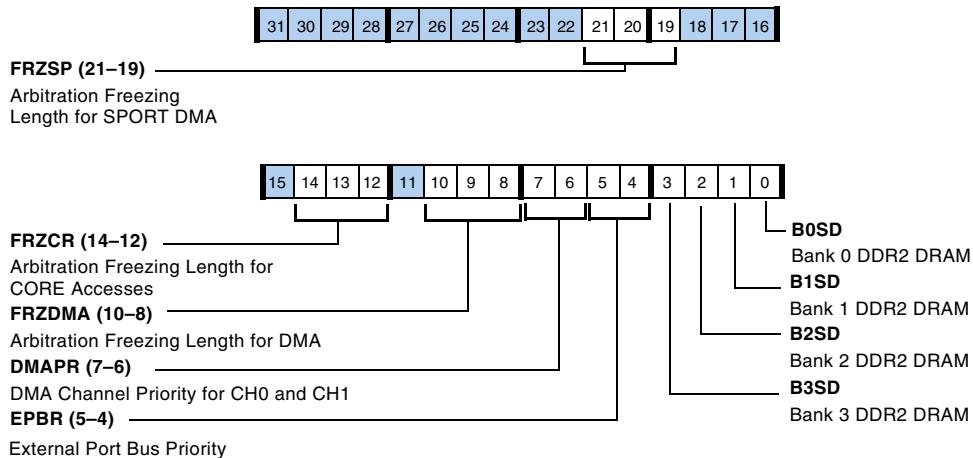


Figure A-6. EPCTL Register

Table A-8. EPCTL Register Bit Descriptions (RW)

Bit	Name	Description
0	B0SD	Select Bank 0 DDR2/SDRAM. 0 = Bank 0 non-DDR2 1 = Bank 0 DDR2
1	B1SD	Select Bank 1 DDR2. 0 = Bank 1 Non-DDR2 1 = Bank 1 DDR2
2	B2SD	Select Bank 2 DDR2 DRAM. 0 = Bank 2 Non-DDR2 1 = Bank 2 DDR2
3	B3SD	Select Bank 3 DDR2 DRAM. 0= Bank 3 Non-DDR2 1= Bank 3 DDR2

ADSP-2146x External Port Registers

Table A-8. EPCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
5–4	EPBR	External Port Bus Priority. 00 = Priority order from highest to lowest is SPORT, external port DMA, core 01 = Priority order from highest to lowest is external port DMA, SPORT, core 10 = Highest priority is core. SPORT and external port DMA are in rotating priority 11 = Rotating priority (default)
7–6	DMAPR	External Port DMA Channel Priority. 00 = Reserved 01 = EP DMA channel 1 high priority 10 = EP DMA channel 0 high priority 11 = Rotating priority (default)
10–8	FRZDMA	Arbitration Freezing Length for DMA. 000 = No Freezing 001 = 4 Accesses 010 = 8 Accesses 011 = 16 Accesses 100 = 32 Accesses 101 = Page size (DDR2 only ¹) 110, 111 = Reserved
11	Reserved	
14–12	FRZCR	Arbitration Freezing Length for CORE Accesses. 000 = No Freezing 001 = 4 Accesses 010 = 8 Accesses 011 = 16 Accesses 100 = 32 Accesses 101 = Page size (DDR2 only ¹) 110, 111 = Reserved
18–15	Reserved	

Table A-8. EPCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
21–19	FRZSP	Arbitration Freezing Length for SPORT DMA. 000 = No Freezing 001 = 4 Accesses 010 = 8 Accesses 011 = 16 Accesses 100 = 32 Accesses 101 = Page size of DDR2 ¹ 110, 111 = Reserved
31–19	Reserved	

1 The EPCTL register automatically reads the DDR2CTL0 page size setting (DDR2CAW), programs just need to program the EPCTL for selecting page size freeze mode.

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-7](#) and described in [Table A-9](#). Note for all AMI timing bit settings, all defined cycles are derived from the DDR2 clock.

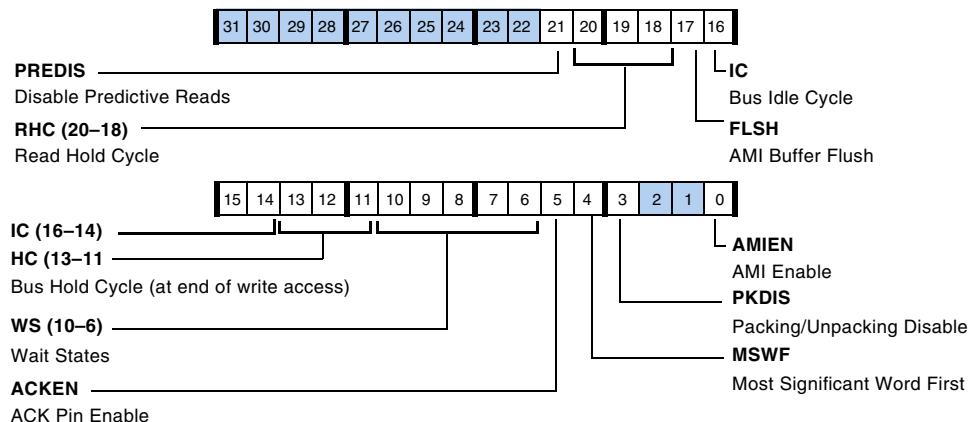


Figure A-7. AMICTLx Registers

ADSP-2146x External Port Registers

Table A-9. AMICTLx Register Bit Descriptions (RW)

Bit	Name	Description
0	AMIEN	AMI Enable. 0 = AMI is disabled 1 = AMI is enabled
2–1	Reserved	
3	PKDIS	Disable Packing/Unpacking. 0 = 8-bit data received packed to 32-bit data. Similarly, 32-bit data to be transmitted is unpacked to four 8-bit data. 1 = 8-bit data received zero-filled, for transmitted data only 8-bit LSB part of the 32-bit data is written to external memory. Note this bit should not be set for bank 0 which is to be used for instruction fetch
4	MSWF	Most Significant Word First. Applicable only with packing disabled (PKDIS=0). 0 = 1st 8-bit word read/write occupies the least significant position in the 32-bit packed word. 1 = 1st 8-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 00010 = wait state = 2 11111 = Wait state = 31
13–11	HC	Bus Hold Cycle at the End of Write Access. 000 = Disable bus hold cycle 001 = Hold address for one external port clock cycle 010 = Hold address for two external port clock cycles

Table A-9. AMICLx Register Bit Descriptions (RW) (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 cycle 010 = Hold address for two cycles
21	PREDIS	Disable Predictive Reads. Default is predictive reads are enabled. For more information, see “Read Optimazition” on page 3-43.
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-8](#) and described in [Table A-10](#).

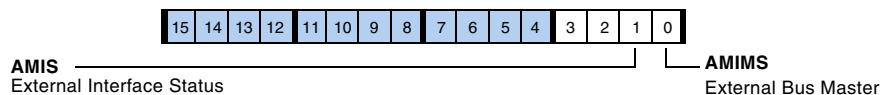


Figure A-8. AMISTAT Register

Table A-10. AMISTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	AMIMS	External Bus Master. 1 = AMI controls the external pins Since the AMI has dedicated pins AMIMS always reads 1.
1	AMIS	External Interface Status. 0 = AMI interface idle 1 = AMI access pending
15–4	Reserved	

DDR2 Registers

The DDR2 controller contains seven memory-mapped control registers. Note that when using the DDR2 registers:

- Programs may write to the DDR2 control registers as long as the controller is not accessing memory devices. Otherwise, the controller responds to any writes to its registers after it finishes any ongoing memory accesses.
- The DDR2 control registers contain sensitive timing parameters and settings for the DDR2. Carefully program these registers with values that are in the operating range of the DDR2 used.
- Values in the reserved fields in these registers must be maintained according to the DDR2 specification. Writing to reserved fields or writing any reserved values in register bits cause the DDR2 to function erroneously.
- Programs must not change prefetch length fields of registers during an ongoing transfer on data buses; otherwise unpredictable behavior may occur.

DDR2 Control Register 0 (DDR2CTL0)

The DDR DDR2CTL0 register includes the programmable parameters associated with the DDR configuration. [Figure A-9](#) and [Table A-11](#) show the corresponding control bit definitions. The FEMRx, FLMR, FDLLCAL, FAR, FPC, SREF_EXIT, DDR2SRF, and DDR2PSS bits are automatically cleared on the next clock edge cycle after they are set.

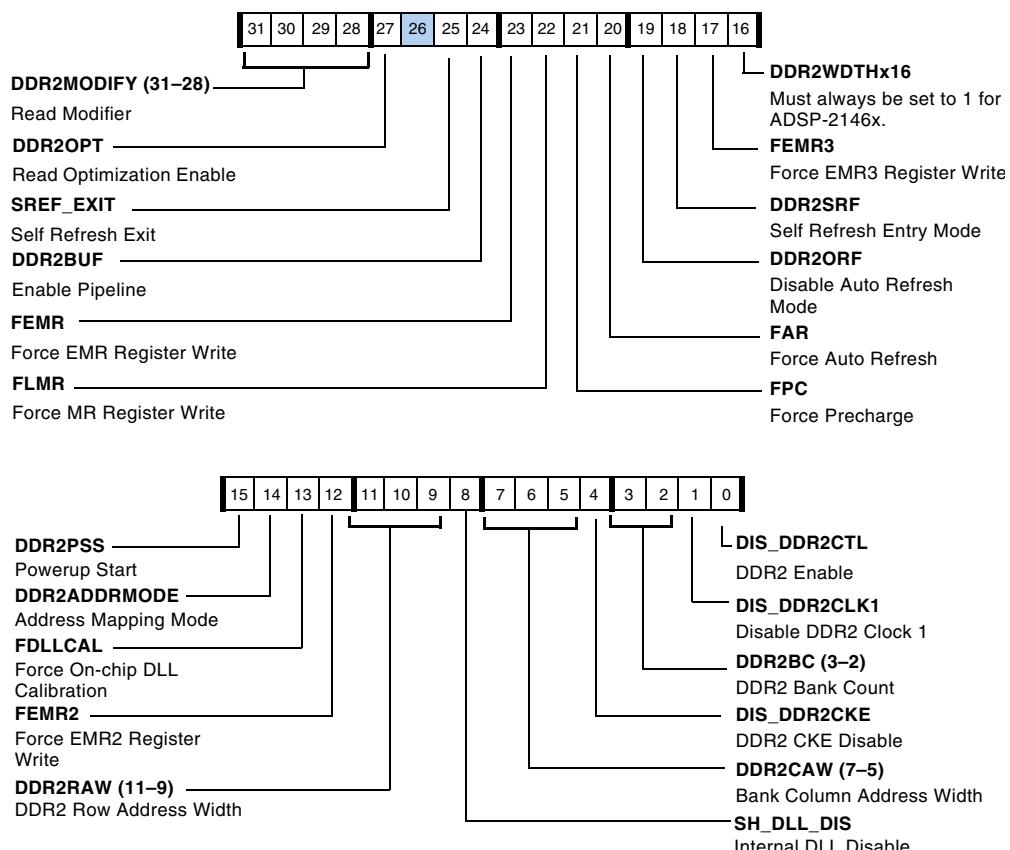


Figure A-9. DDR2CTL0 Register

ADSP-2146x External Port Registers

Table A-11. DDR2CTL0 Register Bit Descriptions (RW)

Bit	Name	Description
0	DIS_DDR2CTL	Disable DDR2 Controller. Enable or disable the DDR2 controller. If the controller is disabled, no accesses to external DDR2 DRAM address spaces occur. All associated control pins (DDR2_RAS, DDR2_CAS, DDR2_WE, DDR2_CS, DDR2_ODT except DDR2_CKE) are in their inactive states and the DDR2 clock is also disabled. 0 = Enable Controller 1 = Disable Controller This bit should not be set when DDR2 interface is active, it can be set in self refresh mode to reduce pad power consumption.
1	DIS_DDR2CLK1	Disable DDR2 Clock 1. Used to disable the 2nd output clock of the controller. By default, both output clocks are driven. 0 = Activate 1 = Disable
3–2	DDR2BC	Bank Count 4 or 8 Bank Device. 00 = Reserved 01 = 4 Bank device 10 = 8 Bank device 11 = Reserved
4	DIS_DDR2CKE	Precharge Power-Down Mode. If set, the DDR2CKE signal is deasserted to bring the DDR2 into precharge power-down mode. Note that memory banks are not refreshed in this mode. 1 = Enter Precharge Power-down Mode 0 = Exit Precharge Power-down Mode
7–5	DDR2CAW	Bank Column Address Width. 000 = Page width 256 001 = Page width 512 010 = Page width 1024 011 = Page width 2048 100 = Page width 4096 Other values are reserved
8	SH_DLL_DIS	SHARC On-Chip DLL Disable. Bypass on-chip DLL. 0 = Enable SHARC DLL0 and 1 1 = Disable SHARC DLL0 and 1

Table A-11. DDR2CTL0 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
11–9	DDR2RAW	Row Address Width. 000 = 8 bits 001 = 9 bits ... 111 = 15 bits
12 (WO)	FEMR2	Force EMR2 Register Write. Forces EMR2 only if the banks are all precharged. 0 = No effect 1 = Force EMR2 register write to DDR2
13 (WO)	FDLLCAL	Force DLL External Bank Calibration. Triggers a DLL calibration sequence to all of the external banks assigned to DDR2 controller in the EPCTL register (BxSD bits). Note for each bank the calibration takes 300 DDR2 cycles. 0 = No effect 1 = Trigger DLL for external bank calibration
14	DDR2ADDRMODE	Select the Address Mapping. This bit selects how the data are stored in the DDR2 memory. 0 = Page interleaving map (consecutive pages/different banks) 1 = Bank interleaving map (consecutive banks)
15 (WO)	DDR2PSS	Power-Up Sequence Start. The power-up sequence is started by setting this bit. Note that the entire power-up sequence takes many cycles to complete. The more external banks assigned, the longer the power-up time. 0 = No effect 1 = Trigger power-up sequence Note that the power-up sequence does NOT require a user access to be executed.
16	DDR2WDTHx16	External Data Path Width. Programs should always set (=1) this bit. 0 = Reserved 1 = 16-bit
17 (WO)	FEMR3	Force EMR3 Register Write. Forces EMR3 only if the banks are all precharged. 0 = No effect 1 = Force EMR3 register write to DDR2

ADSP-2146x External Port Registers

Table A-11. DDR2CTL0 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
18 (WO)	DDR2SRF	Self-Refresh Mode. 0 = No effect 1 = Enters sel-refresh mode
19	DDR2ORF	Auto-Refresh Command. If this bit is set, the auto-refresh command is not issued to the DDR2 memory. This mode allows data streaming connection to FPGA where the refresh is not required. 0 = Auto-refresh command occurs when refresh counter expires. 1 = Auto-refresh not performed
20 (WO)	FARF	Force Auto-Refresh. This bit allows programs to explicitly trigger an auto-refresh command. To use this bit requires that bit 21 is also set, otherwise the DDR2 may crash. 0 = No effect 1 = Force auto-refresh
21 (WO)	FPC	Force Precharge All. This bit allows programs to explicitly trigger a PREA command. 0 = No effect 1 = Force precharge
22 (WO)	FLMR	Force Load Mode Register. Forces MR only if the banks are all precharged. 0 = No effect 1 = Force MR register write to DDR2
23 (WO)	FEMR	Force EMR Register Write. Forces EMR only if the banks are all precharged. 0 = No effect 1 = Force EMR register write to DDR2
24	DDR2BUF	Enable Pipeline. Enabled this bit if the nominal capacitive load is exceeded by connecting DDR2 chips in parallel (4 x IO4). 0 = Disable 1 = External DDR2 control/address buffer enable
25 (WO)	SREF_EXIT	Self-Refresh Exit.

Table A-11. DDR2CTL0 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
27	DDR2OPT	Read Optimization Enable. 0 = Disable read optimization 1 = Enable read optimization
31–28	DDR2MODIFY	Read Modifier (In Optimization Mode). 0000 = Modifier 0 0001 = Modifier 1 ... 1111 = Modifier 15 Note that these bits only are used in SISD mode.

DDR2 Timing Control Register 1 (DDR2CTL1)

The DDR2CTL1 register includes the programmable parameters associated with the DDR access timing. [Figure A-10](#) and [Table A-12](#) show the DDR timing control bit definitions. All the values are defined in terms of number of clock cycles.

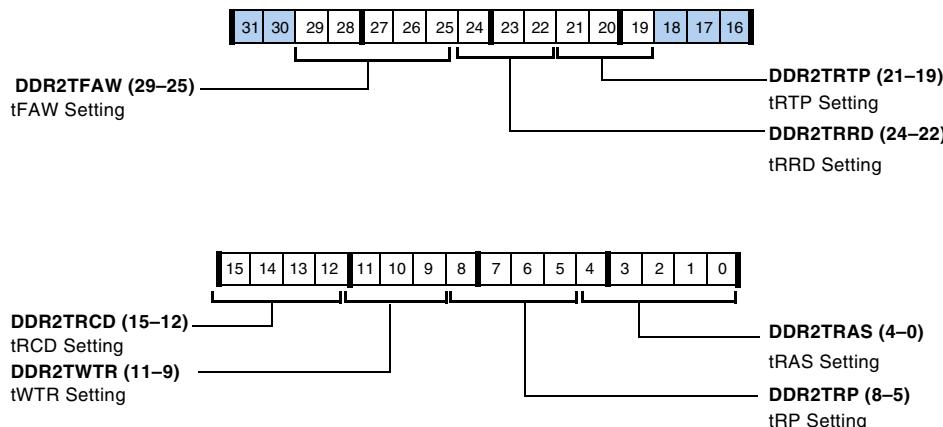


Figure A-10. DDR2CTL1 Register

ADSP-2146x External Port Registers

Table A-12. DDR2CTL1 Register Bit Descriptions (RW)

Bit	Name	Description
4–0	DDR2TRAS	Row Active Time. 00000 = Reserved 00001 = 1 clock cycle 00010 = 2 clock cycles ... 11111 = 31 clock cycles
8–5	DDR2TRP	Row Precharge Time. Note that for 8 banked devices the timing spec becomes $t_{RP} + 1t_{CK}$. 0000 = Reserved 0001 = 1 clock cycle 0010 = 2 clock cycles ... 1111 = 15 clock cycles
11–9	DDR2TWTR	Write to Read Delay. 000 = Reserved 001 = 1 clock cycle 010 = 2 clock cycles ... 111 = 7 clock cycles
15–12	DDR2TRCD	RAS to CAS Delay. 000 = Reserved 001 = 1 clock cycle 010 = 2 clock cycles ... 111 = 7 clock cycles
18–16	Reserved	
21–19	DDR2TRTP	Read to Precharge Delay. 000 = Reserved 001 = 1 clock cycle 010 = 2 clock cycles ... 111 = 7 clock cycles

Table A-12. DDR2CTL1 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
24–22	DDR2TRRD	Row to Row Activation Delay. 000 = Reserved. 001 = 1 clock cycle 010 = 2 clock cycles ... 111 = 7 clock cycles
29–25	DDR2TFAW	Force Activation Window. For 8 banked devices up to 4 banks open in activation window. For 4 banked devices the settings are ignored. 00000 = Reserved 00001 = 1 clock cycle 00010 = 2 clock cycles ... 11111 = 31 clock cycles
31–30	Reserved	

DDR2 Control Register 2 (DDR2CTL2)

Figure A-11 and Table A-13 show the DDR2 control register 2 bit definitions. Values written into this register are loaded into the DDR2 mode register during power up (or when Force LMR bit in the DDR2CTL0 register is set). This register should be initialized before starting the Initialization sequence.



This register's contents should not be changed while DDR2 interface is active. Also whenever this register contents are changed a initialization sequence must be executed to reflect this register contents in to the DDR2 mode register.

ADSP-2146x External Port Registers

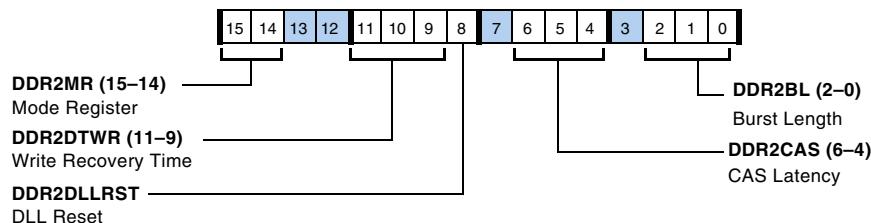


Figure A-11. DDR2CTL2 Register

Table A-13. DDR2CTL2 Register Bit Descriptions (RW)

Bit	Name	Description
2-0	DDR2BL	Burst Length. 010 = BL = 4 All other settings reserved.
3	Reserved	
6-4	DDR2CAS	CAS Latency. 000 = Reserved 001 Reserved 010 2 clock cycles ... 111 7 clock cycles
7	Reserved	
8	DDR2DLLRST	DLL Reset. 0 = Normal 1 = Reset
11-9	DDR2TWR	Write Recovery Time. 000 = Reserved 001 = 2 clock cycle 010 = 3 clock cycles ... 110 = 7 clock cycles 111 = Reserved
12-13	Reserved	
15-14	DDR2MR	Mode Register. Must be set to 00.

DDR2 Control Register 3 (DDR2CTL3)

The DDR2CTL3 register includes the programmable parameters associated with the DDR2 extended mode register (EMR1). [Figure A-12](#) and [Table A-14](#) show the DDR2 control register 3 bit definitions. All the values are defined in terms of number of clock cycles. Values written into this register are loaded into the DDR2 extended mode register during power up (or when Force EMR bit in DDR2CTL0 is set). This register should be initialized before starting the initialization sequence.



This register's contents should not be changed while DDR2 interface is active. Also, whenever this register's contents are changed, an initialization sequence must be executed to reflect this register contents in to the DDR2 extended mode register.

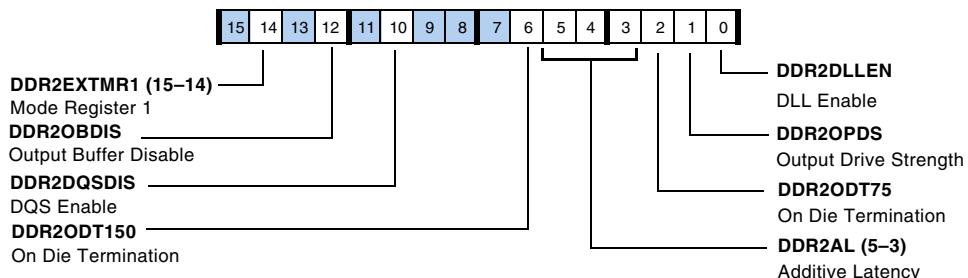


Figure A-12. DDR2CTL3 Register

Table A-14. DDR2CTL3 Register Bit Descriptions (RW)

Bit	Name	Description
0	DDR2DLLEN	DLL Enable. 0 = DLL enabled 1 = DLL disabled
1	DDR2OPDS	Output Drive Strength. 0 = Full strength 1 = Reduced strength

ADSP-2146x External Port Registers

Table A-14. DDR2CTL3 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
5–3	DDR2AL	Additive Latency. Additive latency reduces command bus conflicts to enable commands to be issued more efficiently. Note that the DDR2 controller performance is primary regardless of the AL settings. 000 = 0 clock cycles. 001 = 1 clock cycles. ... 101 = 5 clock cycles. 110, 111 = Reserved. The basic rule for additive latency is defined as: $AL \leq t_{RCDmin}-1$. Example: t_{RCD} is 4 cycles then $AL = 3-0$. If $AL = 4$, t_{RCD} is increased by 1 cycle, increasing overall latency.
6, 2	DDR2ODT	On Die Termination Value. 00 = ODT disabled 01 = 75 ohm 10 = 150 ohm 11 = 50 ohm
9–7	Reserved	
10	DDR2DQSDIS	Differential DQS Disable. 0 = Enable DQS and \overline{DQS} 1 = Disable \overline{DQS}
11	Reserved	
12	DDR2OBDIS	Output Buffer Disable. 0 = Enable 1 = Disable
13	Reserved	
15–14	DDR2EXTMR1	Extended Mode Register 1. Must be set to 01.

DDR2 Control Register 4 (DDR2CTL4)

The DDR2CTL4 register includes the programmable parameters associated with the DDR2 extended mode register 2 (EMR2). [Table A-15](#) shows the DDR2 control register bit definition. All the values are defined in terms of number of clock cycles. Values written into this register are loaded into the DDR2 extended mode register 3 during power up (or when the force EMR bit in the DDR2CTL0 register is set). This register should be initialized before starting the initialization sequence.

-  This register's contents should not be changed while DDR2 interface is active. Also whenever this register contents are changed an initialization sequence must be executed to reflect this register contents in to the DDR2 extended mode register 3.

Table A-15. DDR2CTL4 Register Bit Descriptions (RW)

Bit	Name	Description
13–0	Reserved.	
15–14	DDR2EXTMR2	Extended Mode Register 2. Must be set to 10.

DDR2 Control Register 5 (DDR2CTL5)

The DDR2CTL5 register includes the programmable parameters associated with the DDR2 extended mode register 3 (EMR3). [Table A-16](#) shows the DDR2 control register bit definition. All the values are defined in terms of number of clock cycles. Values written into this register are loaded into the DDR2EMR2 register during power up (or when the Force EMR bit in DDR2CTL0 is set). This register should be initialized before starting the initialization sequence.

ADSP-2146x External Port Registers



This register's contents should not be changed while the DDR2 interface is active. Also, whenever this register's contents are changed an initialization sequence must be executed to reflect this register's contents in the `DDR2EMR2` register.

Table A-16. DDR2CTL5 Register Bit Descriptions (RW)

Bit	Name	Description
13–0	Reserved.	
15–14	DDR2EXTMR3	Extended Mode Register 3. Must be set to 11.

Refresh Rate Control Register (DDR2RRC)

The DDR2 refresh rate control register ([Figure A-13](#) and [Table A-17](#)) provides a flexible mechanism for specifying the auto-refresh timing. The delay (in number of `DDR2CLK` cycles) desired between consecutive refresh counter time-outs must be written to the `RDIV` field. [For more information, see “Refresh Rate” on page 3-33.](#)

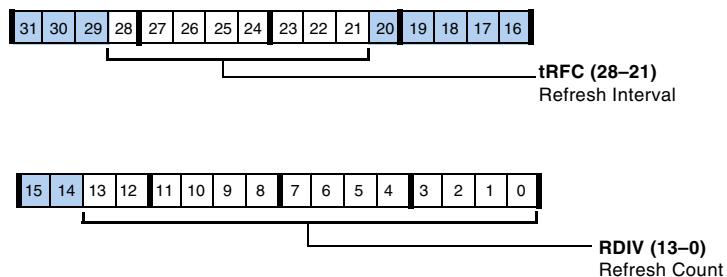


Figure A-13. DDR2RRC Register

Table A-17. DDR2RRC Register Bit Descriptions (RW)

Bit	Name	Description
13–0	RDIV	RDIV value defines the number of clock cycles between two refresh commands.
20–24	Reserved	
28–31	tRFC	Row refresh interval minimum refresh interval in clock cycles. Programmable from 0 to 255.
31–29	Reserved	

Controller Status Register 0 (DDR2STAT0)

The register ([Figure A-14](#) and [Table A-18](#)) provides information on the state of the controller. This information can be used to determine when it is safe to alter DDR2 controller control parameters or as a debug aid.

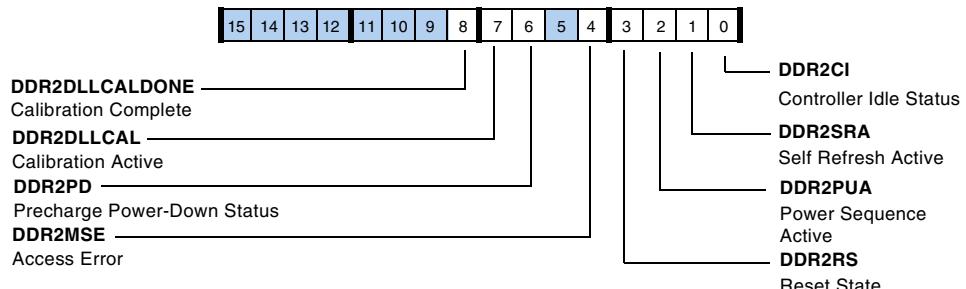


Figure A-14. DDR2STAT0 Register

ADSP-2146x External Port Registers

Table A-18. DDR2STAT0 Register Bit Descriptions (RO)

Bit	Name	Description
0	DDR2CI	Controller Idle Status. 0 = Controller busy performing access or auto-refresh 1 = Controller idle
1	DDR2SRA	Self-Refresh Active. 0 = Not in self-refresh mode 1 = Active
2	DDR2PUA	Power-Up Sequence Active. 0 = DDR2 not in power-up 1 = DDR2 in power-up initialization sequence
3	DDR2RS	DLL Reset. 0 = A power-up to DDR2 has been initialized since last DDR2 controller reset 1 = No power-up sequence occurred since last DDR2 controller reset
4	DDR2MSE	Access Error. 0 = No Error 1 = An access request to DDR2 lost because controller external pins are in disabled state (DIS_DDCTL set) Write a 0 to clear this bit (if set, sticky bit).
5	Reserved	
6	DDR2PD	Precharge Power-Down Status. 0 = Not in precharge power-down 1 = DDR2 in precharge power-down state (DIS_DDR2CKE bit set and DDR2CKE signal deasserted)
7	DDR2DLLCAL	DLL Calibration Status. 0 = Not in DLL calibration sequence 1 = DLL calibration active
8	DDR2DLLCAL DONE	DLL Calibration Complete. 0 = A DLL calibration sequence is not happened since last DDR2 controller reset 1 = A DLL calibration sequence occurred since last DDR2 controller reset The DLL calibration process is performed regardless of whether an external DDR2 bank is assigned or not.
15–9	Reserved	

Controller Status Register 1 (DDR2STAT1)

This register reports the DDR2 bank active/idle status. This register is shown in [Figure A-15](#) and described in [Table A-19](#).

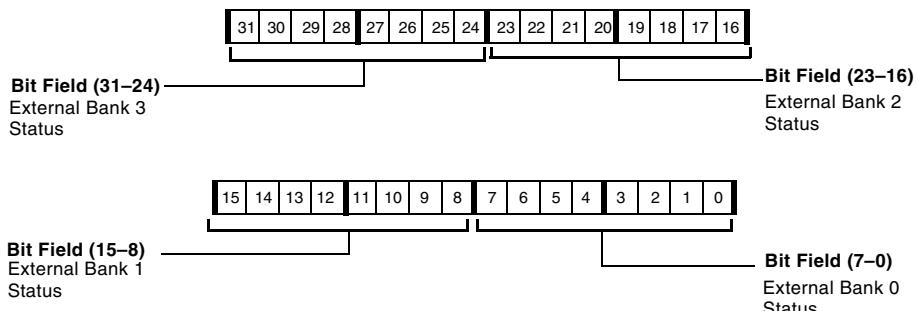


Figure A-15. DDR2STAT1 Register

Table A-19. DDR2STAT1 Register Bit Descriptions (RO)

Bit Field	Field Name	Description
7–0	External Bank 0 Status	<p>External Bank 0 Active/Precharge State.</p> <p>xxxxxx1 = Internal bank 0 in open state xxxxxx0 = Internal bank 0 in precharge state xxxx1x = Internal bank 1 in open state xxxx0x = Internal bank 1 in precharge state ... 1xxxxxx = Internal bank 7 in open state 0xxxxxx = Internal bank 7 in precharge state</p>
15–8	External Bank 1 Status	<p>External Bank 0 Active/Precharge State.</p> <p>xxxxxx1 = Internal bank 0 in open state xxxxxx0 = Internal bank 0 in precharge state xxxx1x = Internal bank 1 in open state xxxx0x = Internal bank 1 in precharge state ... 1xxxxxx = Internal bank 7 in open state 0xxxxxx = Internal bank 7 in precharge state</p>

Table A-19. DDR2STAT1 Register Bit Descriptions (RO) (Cont'd)

Bit Field	Field Name	Description
23–16	External Bank 2 Status	External Bank 0 Active/Precharge State. xxxxxxx1 = Internal bank 0 in open state xxxxxxx0 = Internal bank 0 in precharge state xxxxxx1x = Internal bank 1 in open state xxxxxx0x = Internal bank 1 in precharge state ... 1xxxxxxxx = Internal bank 7 in open state 0xxxxxxxx = Internal bank 7 in precharge state
31–24	External Bank 3 Status	External Bank 0 Active/Precharge State. xxxxxxx1 = Internal bank 0 in open state xxxxxxx0 = Internal bank 0 in precharge state xxxxxx1x = Internal bank 1 in open state xxxxxx0x = Internal bank 1 in precharge state ... 1xxxxxxxx = Internal bank 7 in open state 0xxxxxxxx = Internal bank 7 in precharge state

DLL0 Control Register 1 (DLL0CTL1)

The `DLL0CTL1` register shown in [Figure A-16](#) and described in [Table A-20](#) includes the programmable parameters associated with the DLL0 device. Note that it takes at least 9 core clock cycles to perform a DLL reset.

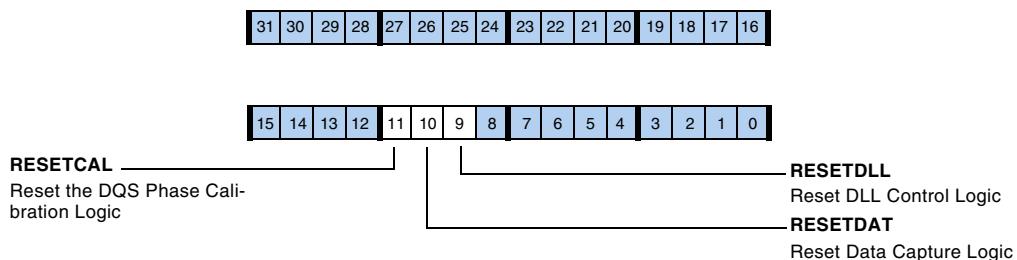


Figure A-16. DLL0CTL1 Register

Table A-20. DLL0CTL1 Register Bit Descriptions (RW)

Bit	Name	Description
8–0	Reserved	
9	RESETDLL	Reset DLL Control Logic. Active high, when active, it resets the DLL control logic only, including the 90 degree DQS shifters. 0 = No effect 1 = Reset DLL0 control logic
10	RESETDAT	Reset Data Capture Logic. Active high, when active, it resets the data capture logic only, including P and N buffers. 0 = No effect 1 = Reset DLL0 data capture logic
11	RESETCAL	Reset DQS Phase Calibration Logic. Active high, when active, it resets the DQS phase calibration logic. 0 = No effect 1 = Reset DLL0 DQS phase calibration logic
31–12	Reserved	

DLL1 Control Register 1 (DLL1CTL1)

The `DLL1CTL1` register shown in [Figure A-17](#) and described in [Table A-21](#) includes the programmable parameters associated with the DLL1 device. Note that it takes at least 9 core clock cycles to perform a DLL reset.

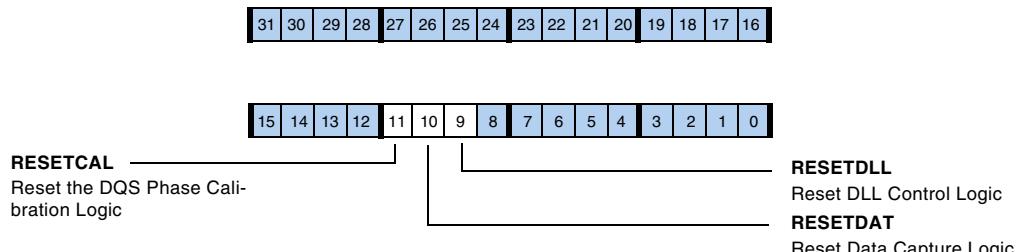


Figure A-17. DLL1CTL1 Register

ADSP-2146x External Port Registers

Table A-21. DLL1CTL1 Register Bit Descriptions (RW)

Bit	Name	Description
8–0	Reserved	
9	RESETDLL	Reset DLL Control Logic. Active high, when active, it resets the DLL control logic only, including the 90 degree DQS shifters. 0 = No effect 1 = Reset DLL1 control logic
10	RESETDAT	Reset Data Capture Logic. Active high, when active, it resets the data capture logic only, including P and N buffers. 0 = No effect 1 = Reset DLL1 reset capture logic
11	RESETCAL	Reset DQS Phase Calibration Logic. Active high, when active, it resets the DQS phase calibration logic. 0 = No effect 1 = Reset DLL1 DQS phase calibration logic
31–12	Reserved	

DLL Status Registers (DLL0STAT0, DLL1STAT0)

The DLL0STAT0 status register indicates the DLL lock status.

Table A-22. DLL0STAT0 Register Bit Descriptions (RW)

Bit	Name	Description
0–29	Reserved	
30	DLL_LOCKED	Reset DLL Control Logic. If this bit is set, indicates that the on-chip DLL for DDR2 controller has locked. Note after reset de-asserted the DLL does automatically lock to the default DDR2 CLK frequency even the DDR2C is not enabled.
31	Reserved	

DDR2 Pad Control Register 0 (DDR2PADCTL0)

The DDR2PADCTL0 register shown in [Figure A-18](#) and described in [Table A-23](#) includes the programmable parameters associated with the DDR2 DATA, DQS and DDR2CLK pads.

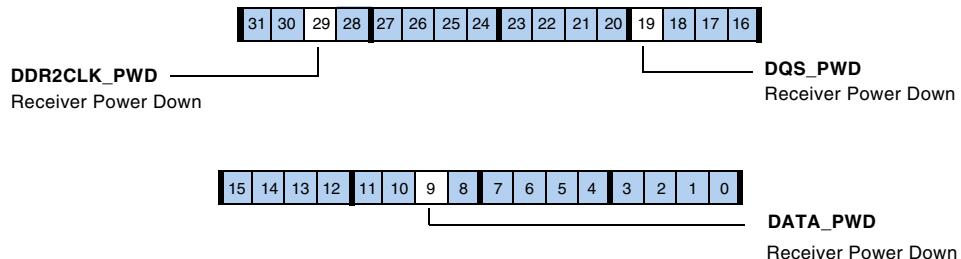


Figure A-18. DDR2PADCTL0 Register

Table A-23. DDR2PADCTL0 Register Bit Descriptions (RW)

Bit	Name	Description
8–0	Reserved	
9	DATA_PWD	Data Pad Receiver Power Down. 0 = Normal mode 1 = Power-down mode
18–10	Reserved	
19	DQS_PWD	DQS Pad Receiver Power Down. 0 = Normal mode 1 = Power-down mode
28–20	Reserved	
29	DDR2CLK_PWD	Clock Pad Receiver Power Down. 0 = Normal mode 1 = Power-down mode
31–30	Reserved	

ADSP-2146x External Port Registers

DDR2 Pad Control Register 1 (DDR2PADCTL1)

The DDR2PADCTL1 register shown in [Figure A-19](#) and described in [Table A-24](#) includes the programmable parameters associated with the DDR2 Command (CS, $\overline{\text{CAS}}$, $\overline{\text{RAS}}$, $\overline{\text{WE}}$, ODT) and Address pad control.

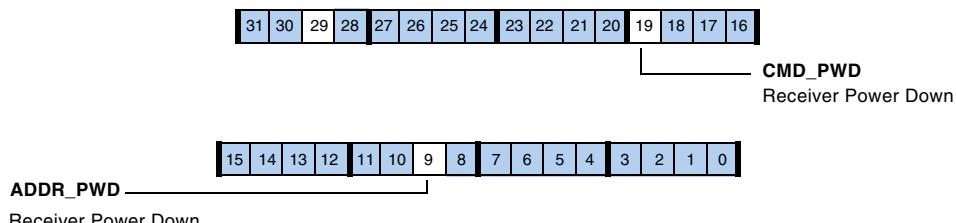


Figure A-19. DDR2PADCTL1 Register

Table A-24. DDR2PADCTL1 Register Bit Descriptions (RW)

Bit	Name	Description
8–0	Reserved	
9	ADDR_PWD	Address Pad Receiver Power Down. 0 = Normal mode 1 = Power-down mode
18–10	Reserved	
19	CMD_PWD	Command Pad Receiver Power Down. 0 = Normal mode 1 = Power-down mode
28–31	Reserved	

ADSP-2147x, ADSP-2148x External Port Registers

The registers in the following sections are specific to the ADSP-2147x and the ADSP-2148x external port and include the external port, the SDRAM controller, and AMI registers.

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-20](#) and described in [Table A-25](#).

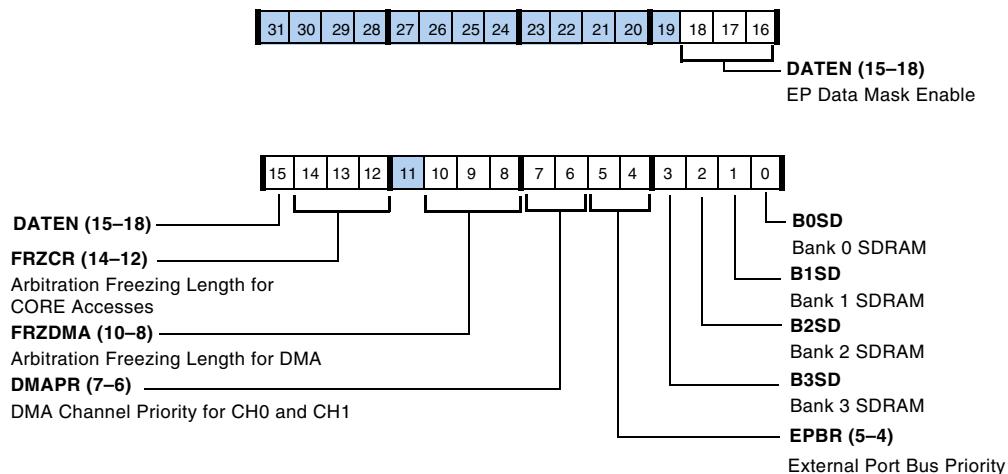


Figure A-20. EPCTL Register

ADSP-2147x, ADSP-2148x External Port Registers

Table A-25. EPCTL Register Bit Descriptions (RW)

Bit	Name	Description
0	B0SD	Select Bank 0 SDRAM. 0 = Bank 0 non-SDRAM 1 = Bank 0 SDRAM
1	B1SD	Select Bank 1 SDRAM. 0 = Bank 1 Non-SDRAM 1 = Bank 1 SDRAM
2	B2SD	Select Bank 2 SDRAM. 0 = Bank 2 Non-SDRAM 1 = Bank 2 SDRAM Note that the $\overline{MS2}$ pin is multiplexed.
3	B3SD	Select Bank 3 SDRAM. 0 = Bank 3 Non-SDRAM 1 = Bank 3 SDRAM Note that the $\overline{MS3}$ pin is multiplexed.
5–4	EPBR	External Port Bus Priority. 00 = Reserved 01 = DMA has high priority 10 = Core has high priority 11 = Rotating priority (default)
7–6	DMAPR	External Port Bus Priority. 00 = Priority order from highest to lowest is SPORT, external port DMA, core 01 = Priority order from highest to lowest is external port DMA, SPORT, core 10 = Highest priority is core. SPORT and external port DMA are in rotating priority 11 = Rotating priority (default)
10–8	FRZDMA	Arbitration Freezing Length for DMA. 000 = No Freezing 001 = 4 Accesses 010 = 8 Accesses 011 = 16 Accesses 100 = 32 Accesses 101 = Page size (SDRAM only) 110, 111 = Reserved
11	Reserved	

Table A-25. EPCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
14–12	FRZCR	Arbitration Freezing Length for CORE Accesses. 000 = No Freezing 001 = 4 Accesses 010 = 8 Accesses 011 = 16 Accesses 100 = 32 Accesses 101 = Page size (SDRAM only ¹) 110, 111 = Reserved
18–15	DATEN	EP Data Mask Enable. In no pack mode of the SDRAM/AMI memory controller, masks those bits of the data lane (DL) with zeros. The data lane is 8 bits. The 16-bit data bus has two data lanes. DATA31–24 is reserved DATA23–16 is reserved DATA15–8 is mapped to DATEN1 DATA7–0 is mapped to DATEN0 For example, if DATEN is 0011, then DL1 and DL01 are masked with zeros.
31–19	Reserved	

- 1 The EPCTL register automatically reads the SDCTL page size setting (SDCAW), programs just need to program the EPCTL for selecting page size freeze mode.

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-21](#) and described in [Table A-26](#). Note for all AMI timing bit settings, all defined cycles are derived from the SDRAM clock.

ADSP-2147x, ADSP-2148x External Port Registers

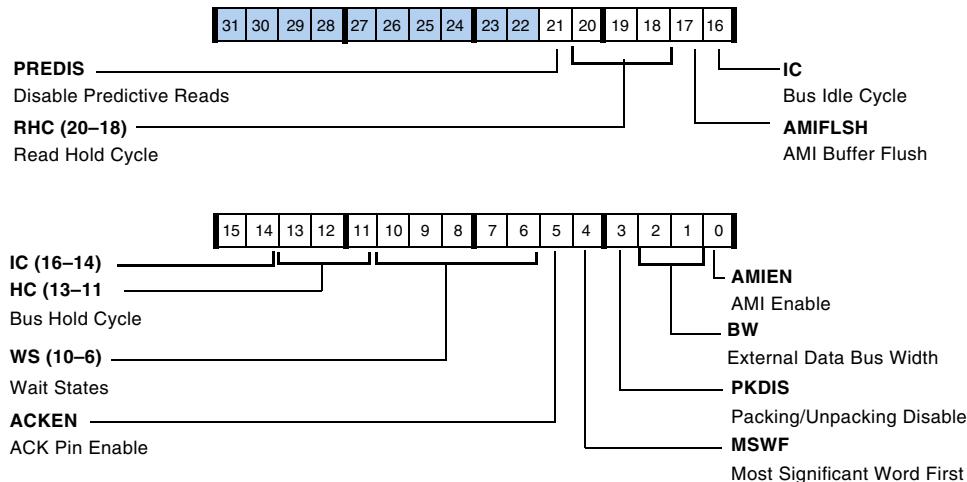


Figure A-21. AMICLx Registers

Table A-26. AMICLx Register Bit Descriptions (RW)

Bit	Name	Description
0	AMIEN	<p>AMIEN. Enables the AMI controller for the dedicated external bank. 0 = AMI is disabled 1 = AMI is enabled</p> <p>To access an external memory bank, the AMIEN bit in the corresponding AMICLx register has to be set. If any of the AMIEN bits are set, then the AMI is enabled and can access memory. However, bank selects can not be driven for that bank whose AMIEN is not set (but read/write strobes can occur).</p> <p>Any access made to a bank whose AMIEN bit is not set occurs at WS = 2 and 8-bit mode without any hold/idle cycles. In any case this access occurs without the bank select and is a void access.</p> <p>Moreover, the AMIEN bit should not be cleared when an access is on-going (when the AMIS bit in the AMISTAT register is set).</p>
2-1	BW	<p>External Data Bus Width. Select between 8-bit and 16-bit. 00 = 8-bit 01 = 16-bit 10, 11 = Reserved</p>

Table A-26. AMICTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
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 four 8-bit data. 1 = 8/16-bit data received zero-filled, for transmitted data only 8-bit LSB part of the 32-bit data is written to external memory. Note this bit should not be set for bank 0 which is to be used for instruction fetch.
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 must be extended by the corresponding devices by pulling ACK low. When ACKEN is set, then the ACK pin is sampled after the wait state value is programmed.
10–6	WS	Wait States. 00000 = Reserved (wait state value of 32 if used) 00001 = wait state=1 (only if ACK input used) 00010 = wait state = 2 11111 = Wait state = 31 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.
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 A bus hold cycle is an inactive bus cycle that the processor automatically generates at the end of a write to allow a longer hold time for address and data. Programs may disable holds, or hold off processing for one or more external port processor cycles. Note the address, data (if a write), and bank select (if in banked external memory) remain unchanged and are driven for one or more cycles after the read or write strobes are deasserted.

ADSP-2147x, ADSP-2148x External Port Registers

Table A-26. AMICLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
16–14	IC	<p>Bus Idle Cycle. Default Idle cycles are inserted whenever read to write in a bank or read to read between two external banks or a read to the SDC happened.</p> <p>000 = 0 cycles, 001 = 1 cycle 010 = 2 cycles, 011 = 3 cycles 100 = 4 cycles, 101 = 5 cycles 110 = 6 cycles, 111 = 7 cycles</p> <p>A bus idle cycle is an inactive bus cycle that the processor automatically generates to avoid data bus driver conflicts. Such a conflict can occur when a device with a long output disable time continues to drive after \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.</p>
17 (WO)	AMIFLSH	<p>AMI Buffer Flush. Flushes both AMIRX and AMITX buffers.</p> <p>0 = Buffer holds the data 1 = Flush the buffer</p>
20–18	RHC	<p>Read Hold Cycle. Controls the delay between two reads.</p> <p>000 = Disable read hold cycle 001 = Hold address for one cycle 010 = Hold address for two cycles</p> <p>A read hold cycle 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.</p>
21	PREDIS	<p>Disable Predictive Reads. Default is predictive reads are enabled. Note this bit is global, if set it does apply to all external banks.</p>
31–22	Reserved	

AMI Status Register (AMISTAT)

This 32-bit register provides status information for the AMI interface and can be read at any time. This register is shown in [Figure A-22](#) and described in [Table A-27](#).

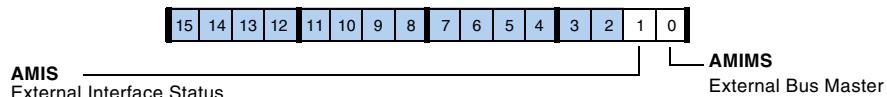


Figure A-22. AMISTAT Register

Table A-27. AMISTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	AMIMS	AMI External Bus Master. 0 = SDRAM Controller controls the external bus 1 = AMI controls the external bus (default) For ADSP-2146x processors bit 0 is always (=1) since the DDR2 and AMI are not shared
1	AMIS	External Interface Status. 0 = AMI interface idle 1 = AMI access pending
15–2	Reserved	

SDRAM Registers

This section provides complete descriptions of the SDRAM controller's memory-mapped registers for SDRAM programming.

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 shown in [Figure A-23](#) and described in [Table A-28](#).

ADSP-2147x, ADSP-2148x External Port Registers

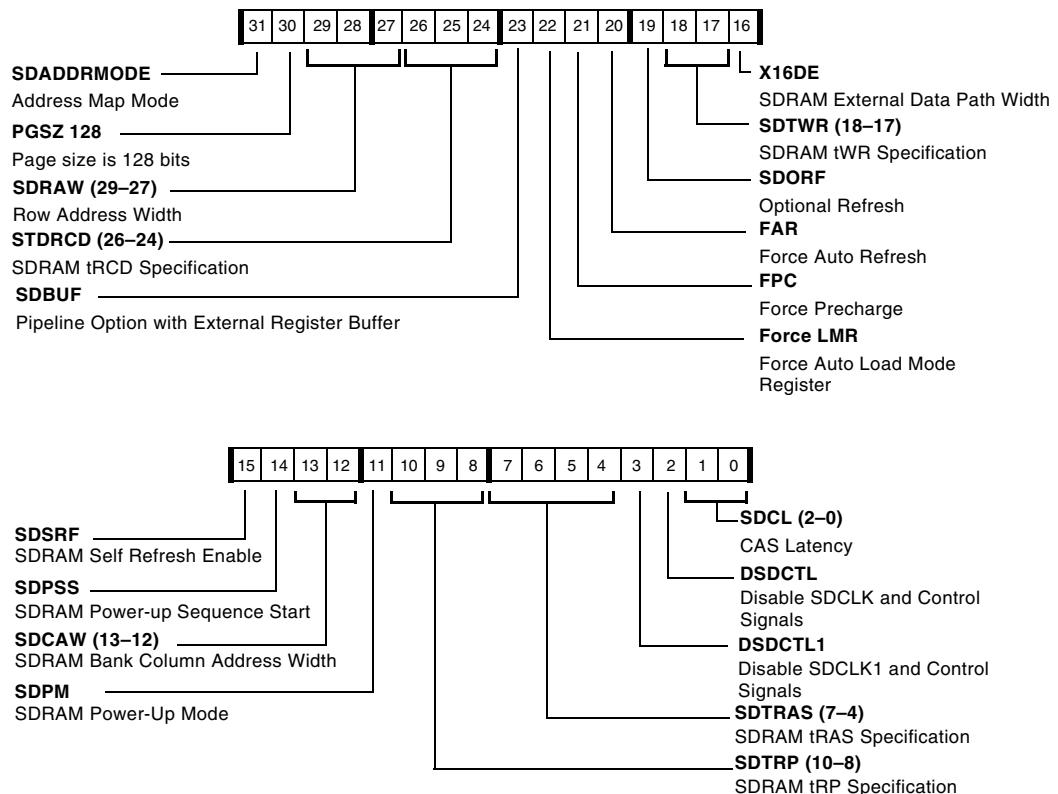


Figure A-23. SDCTL Register

Table A-28. SDCTL Register Bit Descriptions (RW)

Bit	Name	Description
1–0	SDCL	SDRAM CAS Latency. 2–3 SDCLK cycles. The delay in clock cycles between when the SDRAM detects the read command and when it provides the data at its output pins. 00, 01 = Reserved 10 = 2 cycles 11 = 3 cycles A CAS latency of 2 is supported only up to 133 MHz SDCLK.
2	DSDCTL	Disable Controller and Clocks. Used to enable or disable the SDC. 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 When not using SDRAM or when using parts without an external port, systems should disable this bit as early as possible after booting to reduce power consumption.
3	DSDCLK1	Disable Clock SDCLK1. For ADSP-2148x only. 1 = No effect 0 = Disable SDCLK1
7–4	SDTRAS	tRAS Specification. Row Active Open Delay is 1–15 SDCLK cycles. Based on the system clock frequency and the timing specifications of the SDRAM used. Programmed parameters apply to all four banks in the external memory. Refer to the SDRAM data sheet.
10–8	SDTRP	tRP Specification. Row Precharge Delay is 1–8 SDCLK cycles. Based on the system clock frequency and the timing specifications of the SDRAM used. Programmed parameters apply to all four banks in the external memory. Refer to the SDRAM data sheet.
11	SDPM	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. Refer to the SDRAM data sheet.

ADSP-2147x, ADSP-2148x External Port Registers

Table A-28. SDCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
13–12	SDCAW	SDRAM Bank Column Address Width. SDRAM Page Size. 00 = 8 bits 01 = 9 bits 10 = 10 bits 11 = 11 bits
14 (WO)	SDPSS	SDRAM Power-Up Sequence Start. The power-up sequence is triggered by setting this bit. Note that there is a latency for this first access to SDRAM because the SDRAM power-up sequence takes many cycles to complete. 0 = No effect 1 = Enable power-up on next SDRAM access
15 (WO)	SDSRF	Self-Refresh Enable. When the SDSRF bit is set to 1, self-refresh entry command is triggered. Once the SDC completes any active transfers, the SDC executes the sequence of commands to put the SDRAM into self-refresh mode. Any access to the enabled SDRAM bank causes the SDC to trigger a self-refresh exit command.
16	X16DE	SDRAM External Data Path Width. This bit should always be set.
18–17	SDTWR	tWR Specification. Write To Precharge Delay) is 1–3 SDCLK cycles. Based on the system clock frequency and the timing specifications of the SDRAM used. Programmed parameters apply to all four banks in the external memory. Refer to the SDRAM data sheet.
19	SDORF	Optional Auto-Refresh Command. 0 = Auto-refresh occurs when refresh counter expires 1 = Auto-refresh not performed
20 (WO)	FAR	Force Auto-Refresh Command. Performs an auto-refresh immediately. 0 = No effect 1 = Force auto-refresh
21 (WO)	FPC	Force Precharge. Performs a precharge all immediately. 0 = No effect 1 = Force precharge

Table A-28. SDCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
22 (WO)	FMR	Force Load Mode Register Command. This command performs a load mode register command immediately. 0 = No effect 1 = Force MR
23	SDBUF	Pipeline Option with External Register Buffer. 0 = No buffer option 1 = External SDRAM CTL/ADDR control buffer enable
26–24	SDTRCD	SDRAM tRCD Specification. RAS to CAS Delay is = 1–7 SDCLK cycles. Based on the system clock frequency and the timing specifications of the SDRAM used. Programmed parameters apply to all four banks in the external memory. See the SDRAM data sheet.
29–27	SDRAW	Row Address Width. 000=8, 001=9 010=10, 011=11 100=12, 101=13 110=14, 111=15
30	PGSZ 128	Page Size of 128 Words. This bit allows programs to configure the SDC for a page size of 128 words (7 bits) which supports most available 32 Mb SDRAMs. 0 = No effect, page size decided by SDCAW bits. 1 = Page size 128 words. Column width = 7 bits, override CAW settings.
31	SDAD-DRMODE	Select Address Mapping. This bit selects how data is stored in memory. 0 = Bank interleaving 1 = Page interleaving

Control Status Register 0 (SDSTAT0)

The SDRAM control status register provides information on the state of the SDC. This information can be used to determine when it is safe to alter SDC control parameters or as a debug aid. This register is shown in [Figure A-24](#) and described in [Table A-29](#).

ADSP-2147x, ADSP-2148x External Port Registers

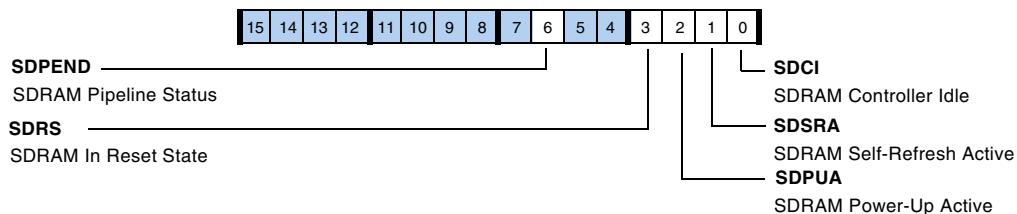


Figure A-24. SDSTAT0 Register

Table A-29. SDSTAT0 Register Bit Descriptions (RO)

Bit	Name	Description
0	SDCI	SDC Idle. This bit is set if the SDC is performing a command or auto-refresh. If no access, this bit is cleared. 0 = SDC idle 1 = SDC access
1	SDSRA	SDC Self-Refresh Mode. If set, controller is in self-refresh mode . 0 = Non self-refresh mode (SDCKE pin high) 1 = Self-refresh mode (SDCKE pin low)
2	SDPUA	SDC Power-Up Active. If set, controller is in power-up mode. 0 = Non power-up mode (SDPSS-bit cleared in SDCTL) 1 = Power-up mode (SDPSS-bit set in SDCTL)
3	SDRS	SDC Reset State. If set, power-up sequence occurred. 0 = No power-up sequence 1 = Power-up sequence occurred
5–4	Reserved	
6	SDPEND	SDC Controller Pipeline Status. 0 = No access pending in controller pipeline 1 = Read/Write access pending in controller pipeline.
15–7	Reserved	

Controller Status Register 1 (SDSTAT1)

This register reports the SDRAM bank active/idle status. This register is shown in [Figure A-25](#) and described in [Table A-30](#).

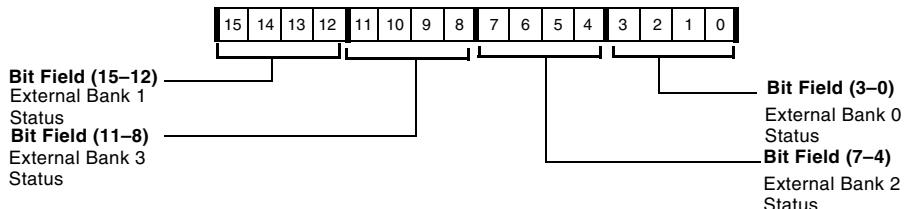


Figure A-25. SDSTAT1 Register

Table A-30. SDSTAT1 Register Bit Descriptions (RO)

Bit Field	Field Name	Description
7–0	External Bank 0 Status	External Bank 0 Active/Precharge State. xxx1 = Internal bank 0 in open state xxx0 = Internal bank 0 in precharge state xx1x = Internal bank 1 in open state xx0x = Internal bank 1 in precharge state ... 1xxx = Internal bank 7 in open state 0xxx = Internal bank 7 in precharge state
15–8	External Bank 1 Status	External Bank 0 Active/Precharge State. xxx1 = Internal bank 0 in open state xxx0 = Internal bank 0 in precharge state xx1x = Internal bank 1 in open state xx0x = Internal bank 1 in precharge state ... 1xxx = Internal bank 7 in open state 0xxx = Internal bank 7 in precharge state

Table A-30. SDSTAT1 Register Bit Descriptions (RO) (Cont'd)

Bit Field	Field Name	Description
23–16	External Bank 2 Status	External Bank 0 Active/Precharge State. xxx1 = Internal bank 0 in open state xxx0 = Internal bank 0 in precharge state xx1x = Internal bank 1 in open state xx0x = Internal bank 1 in precharge state ... 1xxx = Internal bank 7 in open state 0xxx = Internal bank 7 in precharge state
31–24	External Bank 3 Status	External Bank 0 Active/Precharge State. xxx1 = Internal bank 0 in open state xxx0 = Internal bank 0 in precharge state xx1x = Internal bank 1 in open state xx0x = Internal bank 1 in precharge state ... 1xxx = Internal bank 7 in open state 0xxx = Internal bank 7 in precharge state

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 shown in Figure A-26. For information on using the SMODIFY bit see “[SDRAM Read Optimization](#)” on page 3-40.

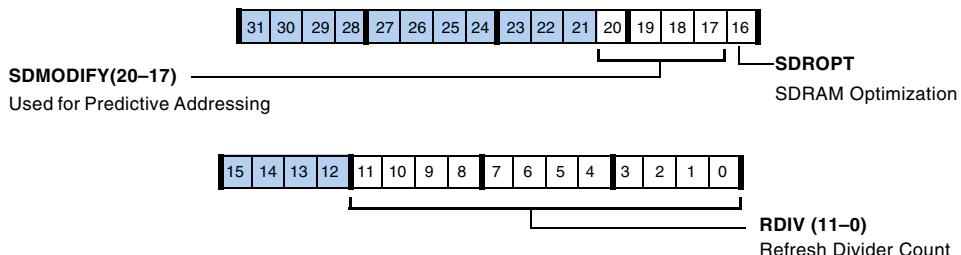


Figure A-26. SDRRC Register

Table A-31. SDRRC Register Bit Descriptions (RW)

Bit	Name	Description
11–0	RDIV	Refresh Divider Count. This 12-bit field defines the number of SDCLK cycles between successive auto-refresh commands. Note that RDIV=0 setting is illegal.
15–12	Reserved	
16	SDROPT	SDRAM Read Optimization. If set (=1) enables read optimization to improve read throughput for core or external port DMA access. 0 = Disabled 1 = Enabled Default setting is 1.
20–17	SDMODIFY	SDRAM Read Modifier. According to SDROPT bit this bit should be set to match the DAG or DMA modifier. 0000 = 0 1111 = 15 Default setting is 1.
31–21	Reserved	

External Port DMA Control Registers (DMACx)

The DMAC0-1 registers control the DMA function of their respective DMA channels. These registers apply to all processors described in this manual and are shown in [Figure A-27](#) and described in [Table A-32](#).

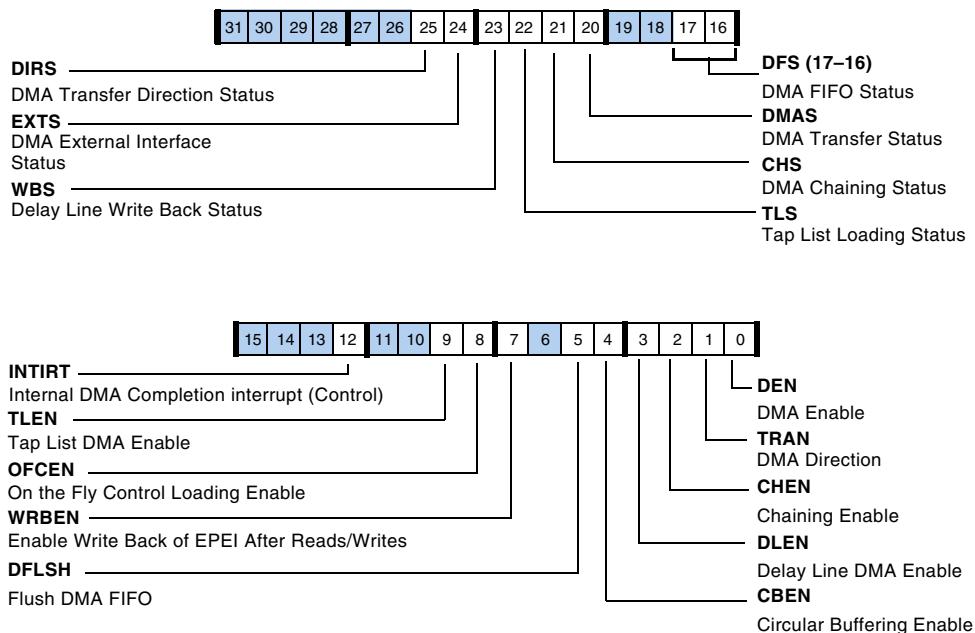


Figure A-27. DMACx Registers

Table A-32. External Port DMA Register Bit Descriptions (RW)

Bit	Name	Description
0	DEN	DMA Enable. 0 = External port channel x DMA is disabled 1 = Enable External port DMA for channel x
1	TRAN	DMA Direction. Determines the DMA data direction. For internal to internal transfers, TRAN must be set. 0 = Write to internal memory (external reads) 1 = Read from internal memory (external writes) Note: If delay line DMA is enabled then the TRAN bit doesn't have any effect. For delay line DMA, transfer direction depends on the state of delay line transfers.
2	CHEN	Enable Chaining. 0 = Chaining disabled 1 = Chaining enabled
3	DLEN	Enable Delay Line DMA. DLEN is applicable only if CHEN=1. 0 = Delay-line DMA disabled 1 = Delay-line DMA enabled
4	CBEN	Circular Buffering Enable. 0 = Disables circular buffering with delay line DMA 1 = Enables circular buffering with delay line DMA Circular buffering can be used with normal DMA as well, if circular buffering is enabled with chaining in normal DMA then ELEP and EBEP should be part of the TCB.
5 (WOC)	DFLSH	Flush DMA FIFO. Clears the DFS bit.
6	Reserved	
7	WRBEN	Enable Write Back of EIEP After Reads/Writes. Write back is automatically enabled for Delay Line DMA. WRBEN is applicable only if CHEN = 1
8	OFCEN	On the Fly Control Loading Enable. The control bits in CPEP register are used to describe the next TCB behavior if OFCEN is set and therefore the DMA controls can be changed from TCB to TCB. 0 = Disables the control bits in CPEP register 1 = Enables the control bits in CPEP register. Note if chaining is enabled with OFCEN bit set then TRAN bit has no effect, and direction is determined by CPD bit in CPEP register.

External Port DMA Control Registers (DMACx)

Table A-32. External Port DMA Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
9	TLEN	Scatter/Gather (Tap List) DMA Enable. 0 = Disables the tap list based scatter/gather DMA 1 = Enables the tap list based scatter/gather DMA
11–10	Reserved	
12	INTIRT	Internal DMA Completion Interrupt (Control). 0 = Interrupt on access completion (internal/external DMA completion depending on external read/write) 1 = Interrupt on internal DMA completion This bit is provided for backward compatibility with older SHARC processors.
15–13	Reserved	
17–16 (RO)	DFS	DMA FIFO Status. 00 = FIFO Empty 01 = FIFO Partially Full 11 = FIFO Full 10 = Reserved
19–18	Reserved	
20 (RO)	DMAS	DMA Transfer Status. 0 = DMA idle 1 = DMA in progress
21 (RO)	CHS	DMA Chaining Status. 0 = DMA chain loading is not active 1 = DMA chain loading is active
22 (RO)	TLS	TAP List Loading Status. 1 = TAP list loading is active 0 = TAP list loading is not active
23 (RO)	WBS	Delay Line Write Pointer Write Back Status. 0 = Write pointer write back is not active 1 = Write pointer write back is active
24 (RO)	EXTS	DMA External Interface Status. 0 = DMA external interface does not have any access pending 1 = DMA external interface has access pending

Table A-32. External Port DMA Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
25 (RO)	DIRS	<p>DMA Transfer Direction Status.</p> <p>0 = DMA direction is external reads 1 = DMA direction is external writes</p> <p>This is useful for delay line DMA where the transfer direction changes with the state of the DMA state machine.</p> <p>For standard DMA, DIRS reflects the state of the TRAN bit.</p>
31–26	Reserved	

Peripheral Registers

The registers in the following sections are used for the peripherals that are not routed through the signal routing units (SRU, SRU2).

Link Port Registers

The following sections describe the link port status and control registers.

Control Register (LCTLx)

[Figure A-28](#) and [Table A-33](#) describe the bit fields within this register.

Peripheral Registers

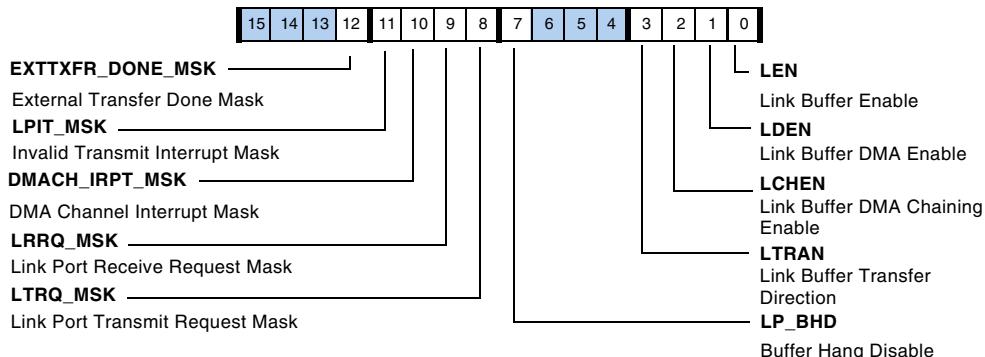


Figure A-28. LCTLx Registers

Table A-33. LCTLx Register Bit Descriptions (RW)

Bit	Name	Description
0	LEN	Link Buffer Enable. Enables (if set, =1) or disables (if cleared, =0) link buffer x (LBUFx). When the processor disables the buffer (LxEN transitions from high to low), the processor clears the corresponding LxSTAT and LxRERR bits.
1	LDEN	Link Buffer DMA Enable. Enables (if set, =1) or disables (if cleared, = 0) DMA transfers link buffer x (LBUFx).
2	LCHEN	Link Buffer DMA Chaining Enable. Enables (if set, =1) or disables (if cleared, =0) DMA chaining link buffer x (LBUFx).
3	LTRAN	Link Buffer Transfer Direction. This bit selects the transfer direction (transmit if set, =1) (receive if cleared, = 0) for link buffer x (LBUFx).
6–4	Reserved	
7	LP_BHD	Buffer Hang Disable. 0 = Core stalls when read from empty receive or write to full transmit buffer attempted 1 = Prevents a core hang.
8	LTRQ_MSK	Link Port Transmit Request Mask. 0 = Mask 1 = Unmask

Table A-33. LCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
9	LRRQ_MSK	Link Port Receive Request Mask. 0 = Mask 1 = Unmask
10	DMACH_IRPT_MSK	DMA Channel Count Interrupt Mask. Must be set to generate interrupt if DMA count is zero and is compatible with traditional SHARC processors. 0 = Mask 1 = Unmask
11	LPIT_MASK	Invalid Transmit Interrupt Mask. 0 = Mask 1 = Unmask
12	EXTTXFR_DONE_MSK	External Transfer Done Interrupt Mask. Valid for core and DMA accesses. If set interrupt is generated when the FIFO is empty. Note if bit 10 is also set for DMA, two interrupts are generated, one for DMA count=0 and one for FIFO empty. 0 = Mask 1 = Unmask
31–13	Reserved	

Status Registers (LSTATx)

Figure A-29 and Table A-34 describe the bit fields within this register.

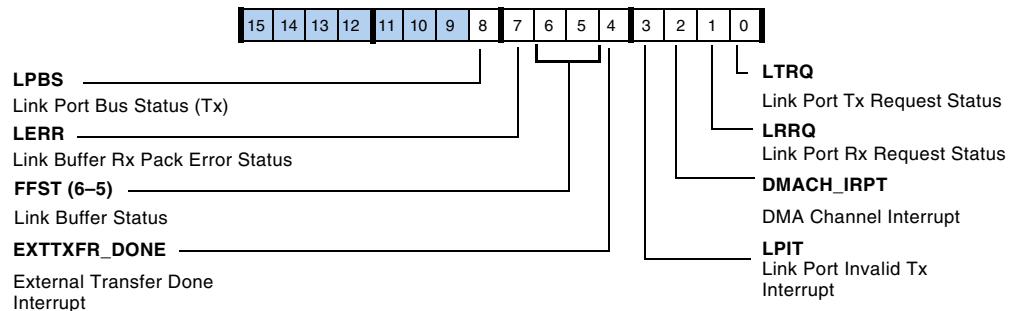


Figure A-29. LSTATx Register

Peripheral Registers

Table A-34. LSTATx Register Bit Descriptions (RO)

Bit	Name	Description
0 (ROC)	LTRQ	Link Port Transmit Request Status.
1 (ROC)	LRRQ	Link Port Receive Request Status.
2 (ROC)	DMACH_IRPT	DMA Channel Count Interrupt.
3 (ROC)	LPIT	Link Port Invalid Transmit Interrupt.
4 (ROC)	EXTTXFR_DONE	External Transfer Done Interrupt.
6–5	FFST	Link Buffer Status. 00 = empty, 01 = reserved, 10 = one word, 11 = full (Cleared when the Link Port is disabled)
7	LERR	Link Buffer Receive Pack Error Status. 0 = Packing complete 1 = Packing incomplete
8	LPBS	Link Port Bus Status (Transmitter). To safely disable linkport transmit operation first poll the FFST bit and second the LPBS bit. 0 = Bus is idle 1 = Bus busy
31–9	Reserved	

Memory-to-Memory Registers

The following DMA related registers are used when performing internal-to-internal DMA through the MTM port.

DMA Control (MTMCTL Register)

The MTMCTL register ([Figure A-30](#)) allows programs to transfer blocks of 64-bit data from one internal memory location to another.

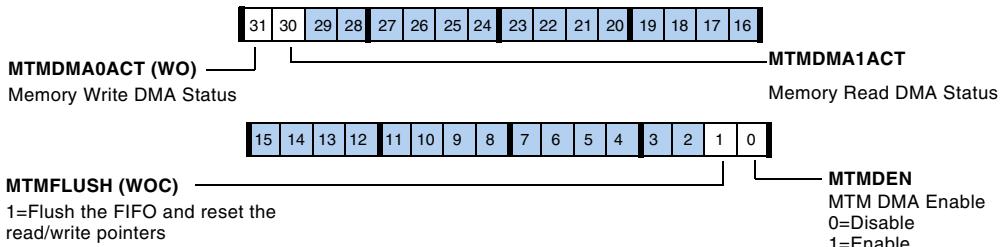


Figure A-30. MTMCTL Register (RW)

Pulse Width Modulation Registers

The following registers control the operation of pulse width modulation on the processor.

Global Control Register (PWMGCTL)

This register enables or disables the four PWM groups, in any combination and provides synchronization across the groups. Note that disable bits have higher priority over the enable bits (bit 1 higher as bit 0 and so on). This 16-bit register is shown in [Figure A-31](#).

For the PWM global control register, the traditional read-modify-write operations to disable the PWM group have changed. The action is to directly write—this simplifies the enable/disable of the PWM groups and can be done with fewer instructions. For example, instead of the following code:

```
ustat3 = dm(PWMGCTL);           /* PWM General Control Register */
bit set ustat3 PWM_DIS0;        /* disables group 0 */
dm(PWMGCTL) = ustat3;
```

Use:

```
ustat3 = PWM_DIS0;
dm(PWMGCTL) = ustat3;
```

Peripheral Registers

Writes to the enable and disable bit-pairs for a PWM group works as follows.

PWM_DISx = 0, PWM_ENx = 0 – No action

PWM_DISx = 0, PWM_ENx = 1 – Enable the PWM group

PWM_DISx = 1, PWM_ENx = x – Disable the PWM group

For reads, the interpretation is as follows.

PWM_DISx = 0, PWM_ENx = 0 – PWM group is disabled

PWM_DISx = 1, PWM_ENx = 1 – PWM group is enabled

Any other read combination is not possible. Reads of the PWMGCTL register returns the enable status on both the enable and disable bits.

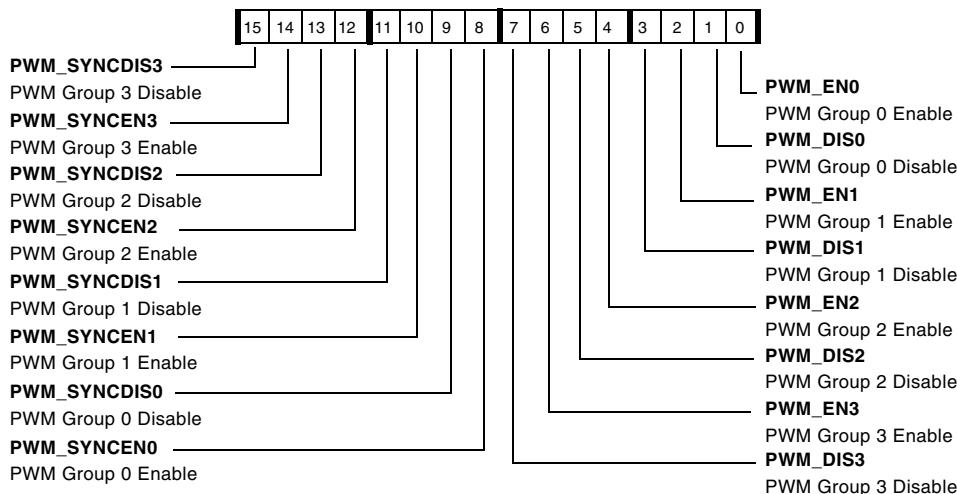


Figure A-31. PWMGCTL Register

Table A-35. PWMGCTL Register Bit Descriptions (RW)

Bit	Name	Function
0, 2, 4, 6	PWM_ENx0	PWM Group x Enable
1, 3, 5, 7	PWM_DISx	PWM Group x Disable

Table A-35. PWMGCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Function
8, 10, 12, 14	PWM_SYNCENx	PWM Group x Enable
9, 11, 13, 15	PWM_SYNCDISx	PWM Group xDisable

Global Status Register (PWMGSTAT)

This register provides the status of each PWM group (Table A-36). The status bits are set depending on the IRQEN bit. The ISR needs to write one to clear the status bits.

Table A-36. PWMGSTAT Register Bit Descriptions (W1C)

Bit	Name	Function
0	PWM_STAT0	PWM Group 0 Period Completion Status
1	PWM_STAT1	PWM Group 1 Period Completion Status
2	PWM_STAT2	PWM Group 2 Period Completion Status
3	PWM_STAT3	PWM Group 3 Period Completion Status
15–4	Reserved	

Control Register (PWMCTLx)

These registers, described in Table A-37, are used to set the operating modes of each PWM block. They also allow programs to disable interrupts from individual groups.

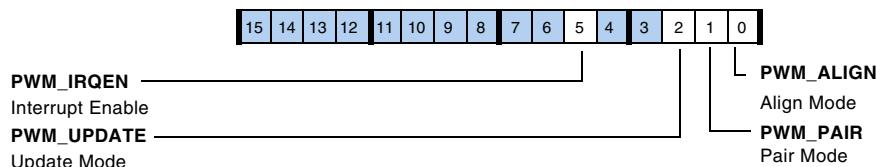


Figure A-32. PWMCTLx Register

Peripheral Registers

Table A-37. PWMCTLx Register Bit Descriptions (RW)

Bit	Name	Description
0	PWM_ALIGN	Align Mode. 0 = Edge-aligned. The PWM waveform is left-justified in the period window. 1 = Center-aligned. The PWM waveform is symmetrical.
1	PWM_PAIR	Pair Mode. 0 = Non-paired mode. The PWM generates independent signals (for example xH, xL) 1 = Paired mode. The PWM generates the complementary signal from the high side output (xL=/xH).
2	PWM_UPDATE	Update Mode. 0 = Single update mode. The duty cycle values are programmable only once per PWM period. The resulting PWM patterns are symmetrical about the mid-point of the PWM period. 1 = Double update mode. A second update of the PWM registers is implemented at the mid-point of the PWM period. PWM_UPDATE mode has only effect for center aligned mode (PWM_ALIGN=1).
4–3	Reserved	
5	PWM_IRQEN	Enable PWM Interrupts. 0 = Interrupts not enabled 1 = Interrupts enabled
15–6	Reserved	

Status Registers (PWMSTATx)

These 16-bit registers, described in [Table A-38](#), report the status of the phase and mode for each PWM group.

Table A-38. PWMSTATx Register Bit Descriptions (RO)

Bit	Name	Description
0	PWM_PHASE	PWM Phase Status. Set during center aligned mode in the second half of each PWM period. Allows programs to determine the particular half-cycle (first or second) during PWM interrupt service routine, if required. 0 = First half 1 = Second half (default) In edge aligned mode this bit is always set.
1	Reserved	
2	PWM_PAIRSTAT	PWM Paired Mode Status. 0 = Inactive paired mode 1 = Active paired mode
15–3	Reserved	

Output Disable Registers (PWMSEGx)

These 16-bit registers, described in [Table A-39](#), control the output signals of the four PWM groups. The output signals are enabled by default.

Table A-39. PWMSEGx Register Bit Descriptions (RW)

Bit	Name	Description
0	PWM_BH	Channel B High Disable. Enables or disables the channel B output signal. 0 = Enable 1 = Disable
1	PWM_BL	Channel B Low Disable. Enables or disables the channel B output signal. 0 = Enable 1 = Disable

Peripheral Registers

Table A-39. PWMSEGx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
2	PWM_AH	Channel A High Disable. Enables or disables the channel A output signal. 0 = Enable 1 = Disable
3	PWM_AL	Channel A Low Disable. Enables or disables the channel A output signal. 0 = Enable 1 = Disable
4	BHBL_XOVR	Crossover Enable for BH/BL Pair. 0 = Disable 1 = Enable
5	AHBL_XOVR	Crossover Enable for AH/AL Pair. 0 = Disable 1 = Enable
15–6	Reserved	

Polarity Select Registers (PWMPOLx)

These 16-bit registers, described in [Table A-40](#), control the polarity of the four PWM groups which can be set to either active high or active low. Note that bit 1 has priority over bit 0, bit 3 over bit 2 and so on. In paired mode, it is expected to maintain polarity coherency by setting the same polarity for both the high and low side of a PWM pair.

Table A-40. PWMPOLx Register Bit Descriptions (RW)

Bit	Name	Description
0	PWM_POL1AL	Channel AL Polarity 1. 0 = Channel AL polarity 0 1 = Channel AL polarity 1 (default)
1	PWM_POL0AL	Channel AL Polarity 0. 0 = Channel AL polarity 0 1 = Channel AL polarity 1 (default)

Table A-40. PWMPOLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
2	PWM_POL1AH	Channel AH Polarity 1. 0 = Channel AH polarity 0 1 = Channel AH polarity 1 (default)
3	PWM_POL0AH	Channel AH Polarity 0. 0 = Channel AH polarity 0 1 = Channel AH polarity 1 (default)
4	PWM_POL1BL	Channel BL Polarity 1. 0 = Channel AL polarity 0 1 = Channel AL polarity 1 (default)
5	PWM_POL0BL	Channel BL Polarity 0. 0 = Channel AL polarity 0 1 = Channel AL polarity 1 (default)
6	PWM_POL1BH	Channel BH Polarity 1. 0 = Channel BH polarity 0 1 = Channel BH polarity 1 (default)
7	PWM_POL0BH	Channel BH Polarity 0. 0 = Channel BH polarity 0 1 = Channel BH polarity 1 (default)
15–8	Reserved	

Period Registers (PWMPERIODx)

These 16-bit RW registers control the unsigned period of the four PWM groups. This register is double buffered for double update mode. A change in one half cycle of PWM switching period only takes effect in the next half period.

Duty Cycle High Side Registers (PWMAx, PWMBx)

The 16-bit duty-cycle control registers (RW) directly control the A/B (two's-complement) duty cycles of the two pairs of PWM signals.

Peripheral Registers

Duty Cycle Low Side Registers (PWMA_{Lx}, PWMB_{Lx})

The 16-bit duty-cycle control registers (RW) directly control the AL/BL duty cycles (two's-complement) of the non-paired PWM signals. These can be different from the AH/BH cycles.

Dead Time Registers (PWMDT_x)

These 16-bit RW registers set up a short time delay (10-bit, unsigned) between turning off one PWM signal and turning on its complementary signal.

Debug Status Registers (PWMDBG_x)

These 16-bit registers aid in software debug activities.

Table A-41. PWMDBG_x Register Bit Descriptions (RO)

Bit	Name	Function
0	PWM_AL	Channel A Low Output Signal for S/W Observation
1	PWM_AH	Channel A High Output Signal for S/W Observation
2	PWM_BL	Channel B Low Output Signal for S/W Observation
3	PWM_BH	Channel B High Output Signal for S/W Observation
15:4	Reserved	

FFT Accelerator Registers

The following sections describe the registers used to program and debug the FFT accelerator.

General Control Register (FFTCTL1)

The global control register (FFTCTL1) shown in [Figure A-33](#) and described in [Table A-41](#) is used to enable, start, and reset the FFT module. It is also used to enable DMA and debug operation.

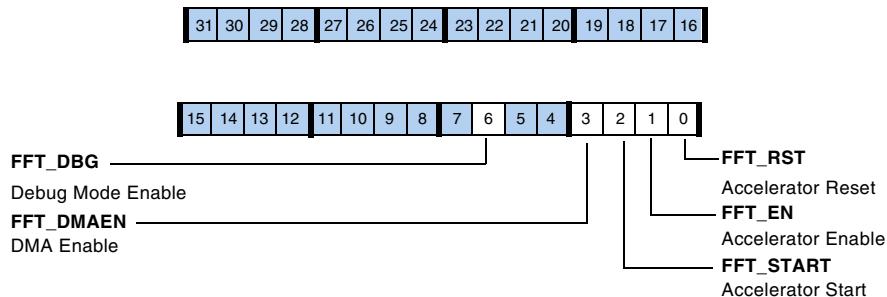


Figure A-33. FFTCTL1 Register

Table A-42. FFTCTL1 Register Bit Descriptions (RW)

Bits	Name	Description
0	FFT_RST	Reset Accelerator. Setting this bit puts the accelerator into reset mode. Explicit clearing of this bit is necessary to take the accelerator out of reset. 0 = Normal operation 1 = Reset
1	FFT_EN	Accelerator Enable. 0 = Disable 1 = Enable
2	FFT_START	Start Accelerator. 0 = Idle 1 = Start
3	FFT_DMAEN	DMA Enable. 0 = Disable 1 = Enable
5–4	Reserved	

Peripheral Registers

Table A-42. FFTCTL1 Register Bit Descriptions (RW) (Cont'd)

Bits	Name	Description
6	FFT_DBG	Debug Mode Enable. 0 = Disable 1 = Enable
31–7	Reserved	

Control Register (FFTCTL2)

The FFT control register, shown in [Figure A-34](#) and described in [Table A-43](#), is used to set up individual FFT parameters (such as length) and how the module process the FFT, such as data packing.

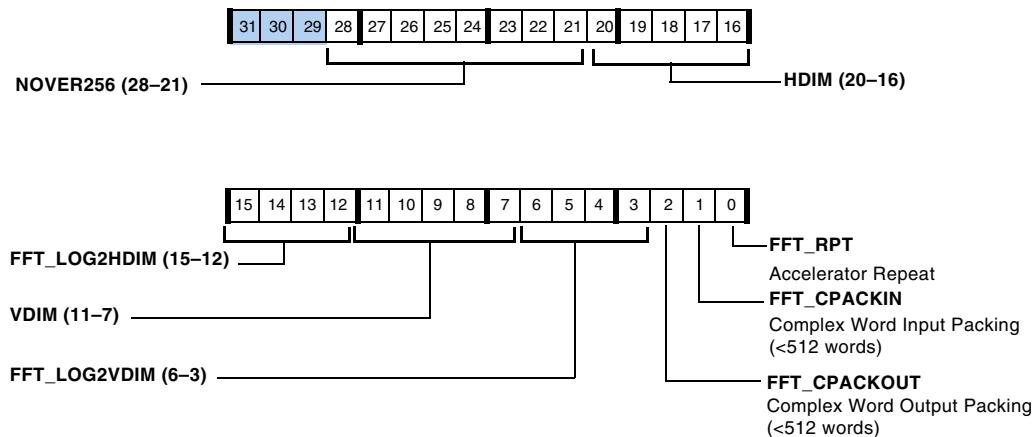


Figure A-34. FFTCTL2 Register

Table A-43. FFTCTL2 Register Bit Descriptions (RW)

Bits	Name	Description
0	FFT_RPT	Accelerator Repeat. If this bit is set and the program needs to change the parameters of the FFT (such as length), first clear the FFT_EN and FFT_START bits in the FFTCTL1. Next change the FFT parameters as needed. Then set FFT_EN and, finally, set FFT_START. The accelerator is in READING mode and is ready to read in data/twiddles and compute FFT with the newly programmed parameters. 0 = Go to idle when done 1 = Repeat when done
1	FFT_CPACKIN	Complex Word Input Packing for <512 Words. 0 = No packing, first all reals, then all imag are received by the accelerator 1 = Complex numbers are packed Real/Imag. For >256 words, this bit is always set.
2	FFT_CPACKOUT	Complex Word Output Packing for <512 Words. 0 = No packing, first all reals, then all imag are sent by the accelerator 1 = Complex numbers are packed Real/Imag. For >256 words, this bit is always set.
6–3	FFT_LOG2VDIM	Log2 (VDIM).
11–7	VDIM	VDIM/16. Vertical Column Dimension of the FFT computation matrix V = 16, VDIM = 1 V = 32, VDIM = 2 V = 64, VDIM = 3 V = 128, VDIM = 4 V = 256, VDIM = 5
15–12	FFT_LOG2HDIM	Log2 (HDIM).
20–16	HDIM	HDIM/16. Horizontal Column Dimension of the FFT computation matrix. For small FFTs, (<512 points) only VDIM required HDIM = 0.
28–21	NOVER256	N/256 = HDIM × VDIM (used for large FFTs)
31–29	Reserved	

Peripheral Registers

Multiplier Status Register (FFTMACSTAT)

The FFT_MACSTAT register, described in [Table A-44](#), can be written only in debug mode. The status bits are sticky and are cleared when read.

Table A-44. FFT_MACSTAT Register Bit Descriptions (ROC)

Bits	Name	Description
0	FFT_NAN	Bits 3–0 follow the IEEE STD for floating point numbers.
1	FFT_DENORM	
2	FFT_OVR	
3	FFT_UDR	
31–4	Reserved	

DMA Status Register

The bits in the status register, described in [Table A-45](#) report DMA status information.

Table A-45. FFTDMASTAT Register Bit Descriptions (RO)

Bits	Name	Description
0	ICPLD	Input Chain Pointer Loading
1	IDMASTAT	Input DMA in Progress
2 (ROC)	IDMACHIRPT	Input DMA Channel Interrupt
3	OCPLD	Output Chain Pointer Loading
4	ODMASTAT	Output DMA in Progress
5 (ROC)	ODMACHIRPT	Output DMA Channel Interrupt
31–6	Reserved	

Debug Registers (FFTADDR, FFTDDATA)

Bits 31–0 is the FFT_DDATA register correspond to the data to be read or written. When a data write is performed first this register is loaded with data which needs to be written, then the FFT_DADDRESS register is loaded with the write address of the location. Note that these registers should be written/read only in debug mode. In [Table A-46](#), A is a meaningful address bit.

Table A-46. DADDRESS Register Bit Descriptions (RW)

Bits	Name	Description
12–0	ADDRESS	<p>Address Bit. Access to local memory requires debug mode. The MSB bits of the address decode the memory location.</p> <p>000AAAAAAAAA = read data memory (2^{10})</p> <p>100AAAAAAAAA = write data memory (2^{10})</p> <p>010xAFFFFFFF = read coefficient memory (2^9)</p> <p>110xAFFFFFFF = write coefficient memory (2^9)</p> <p>A = valid address bits</p>
31–13	Reserved	

FIR Accelerator Registers

The following sections describe the registers used to program and debug the FIR accelerator.

Global Control Register (FIRCTL1)

The FIRCTL1 register, shown in [Figure A-35](#) and described in [Table A-47](#), is used to configure the global parameters for the accelerator. These include the number of channels, channel auto iterate, DMA enable, and accelerator enable.

Peripheral Registers

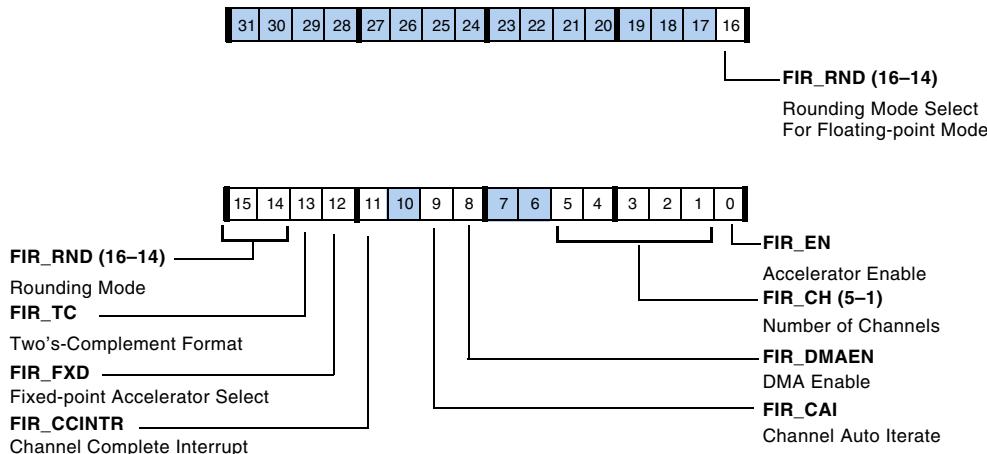


Figure A-35. FIRCTL1 Register

Table A-47. FIRCTL1 Register Bit Descriptions (RW)

Bits	Name	Description
0	FIR_EN	FIR Enable. 0 = FIR disabled 1 = FIR enabled
5–1	FIR_CH32–1	Number of Channels. Programmable between 0–31 0 = FIR_CH1 31 = FIR_CH32
7–6	Reserved	
8	FIR_DMAEN	DMA Enable. 0 = DMA disabled 1 = DMA enabled
9	FIR_CAI	Channel Auto Iterate. 0 = Processing stops once all channels are over 1 = Moves to first channel and continues processing in a loop when all channels are over
10	Reserved	

Table A-47. FIRCTL1 Register Bit Descriptions (RW) (Cont'd)

Bits	Name	Description
11	FIR_CCINTR	Channel Complete Interrupt. 0 = Interrupt is generated only when all channels are done (default) 1= Interrupt is generated after each channel is done
12	FIR_FXD	Fixed-Point Accelerator Select. 0 = 32-bit IEEE floating-point 1 = 32-bit fixed point
13	FIR_TC	Two's-Complement Format Input Select For Fixed-Point Mode. 0 = Unsigned integer 1 = Signed intiger
16–14	FIR_RND	Rounding Mode Select For Floating-Point Mode. 000 = IEEE round to nearest (even) 001 = IEEE round to zero 010 = IEEE round to +ve infinity 011 = IEEE round to -ve infinity 100 = Round to nearest Up 101 = Round away from zero 110 = Reserved 111 = Reserved
31–17	Reserved	

Channel Control Register (FIRCTL2)

The FIRCTL2 register, shown in [Figure A-36](#) and described in [Table A-48](#), is used to configure the channel specific parameters such as filter TAP length, window size, sample rate conversion, up/down sampling and ratio.

Peripheral Registers

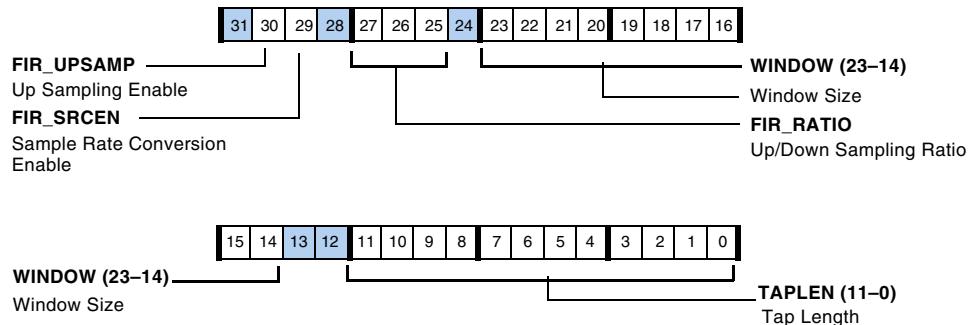


Figure A-36. FIRCTL2 Register

Table A-48. FIRCTL2 Register Bit Descriptions (RW)

Bits	Name	Description
11–0	TAPLEN	Tap Length. Programmable between 0–4095 Tap Length = TAPLEN + 1
13–12	Reserved	
23–14	WINDOW	Window Size. Programmable between 0–1023 Window Size = WINDOW + 1. A window size of 1 corresponds to sample based operation and the maximum window size is 1024.
24	Reserved	
27–25	FIR_RATIO	UP/DOWN Sampling Ratio. Sampling Ratio = RATIO + 1
28	Reserved	
29	FIR_SRCEN	Sample Rate Conversion Enable. 0 = Disabled 1 = Enabled
30	FIR_UPSAMP	Up Sampling Enable. 0 = Down Sampling 1 = Up sampling
31	Reserved	

FIR MAC Status Register (FIRMACSTAT)

This register, shown in [Figure A-37](#) and described in [Table A-49](#), provides the status of MAC operations. The status of all four multipliers/adders are available separately for programs to poll. In fixed-point mode only the ARI_x bits are used (all other bits are reserved).

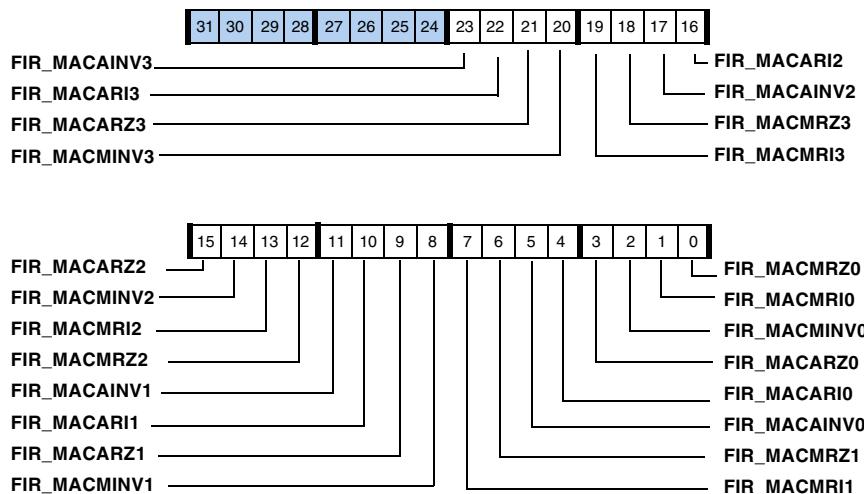


Figure A-37. FIRMACSTAT Register

Table A-49. FIRMACSTAT Register Bit Descriptions (RO)

Bits	Name	Description
0	FIR_MACMRZ0	Multiplier Result Zero. Set if multiplier 0 results is zero.
1	FIR_MACMRI0	Multiplier Result Infinity. Set if multiplier 0 results is infinity.
2	FIR_MACINV0	Multiply Invalid. Set if multiplier 0 multiply operation is invalid.
3	FIR_MACARZ0	Adder Result Zero. Set if a adder 0 results is zero.
4	FIR_MACARIO	Adder Result Infinity. Set if adder 0 results is infinity. Indicates overflow in fixed-point mode.
5	FIR_MACAINV0	Addition Invalid. Set if a adder 0 addition is invalid.

Peripheral Registers

Table A-49. FIRMACSTAT Register Bit Descriptions (RO) (Cont'd)

Bits	Name	Description
6	FIR_MACMRZ1	Multiplier Result Zero. Set if multiplier 1 results is zero.
7	FIR_MACMRI1	Multiplier Result Infinity. Set if multiplier 1 results is infinity.
8	FIR_MACMINV1	Multiply Invalid. Set if multiplier 1 multiply operation is invalid.
9	FIR_MACARZ1	Adder Result Zero. Set if a adder 1 results is zero.
10	FIR_MACARI1	Adder Result Infinity. Set if adder 1 results is infinity. Indicates overflow in fixed-point mode.
11	FIR_MACAINV1	Addition Invalid. Set if a adder 1 addition is invalid.
12	FIR_MACMRZ2	Multiplier Result Zero. Set if multiplier 2 results is zero.
13	FIR_MACMRI2	Multiplier Result Infinity. Set if multiplier 2 results is infinity.
14	FIR_MACMINV2	Multiply Invalid. Set if multiplier 2 multiply operation is invalid.
15	FIR_MACARZ2	Adder Result Zero. Set if a adder 2 results is zero.
16	FIR_MACARI2	Adder Result Infinity. Set if adder 2 results is infinity. Indicates overflow in fixed-point mode.
17	FIR_MACAINV2	Addition Invalid. Set if a adder 2 addition is invalid.
18	FIR_MACMRZ3	Multiplier Result Zero. Set if multiplier 3 results is zero.
19	FIR_MACMRI3	Multiplier Result Infinity. Set if multiplier 3 results is infinity.
20	FIR_MACMINV3	Multiply Invalid. Set if multiplier 3 multiply operation is invalid.
21	FIR_MACARZ3	Adder Result Zero. Set if a adder 3 results is zero.
22	FIR_MACARI3	Adder Result Infinity. Set if adder 3 results is infinity. Indicates overflow in fixed-point mode.
23	FIR_MACAINV3	Addition Invalid. Set if a adder 3 addition is invalid.
31–24	Reserved	

FIR DMA Status Register (FIRDMASTAT)

The information provided by this register, shown in [Figure A-38](#) and described in [Table A-50](#), are, chain pointer loading, coefficient DMA, data preload DMA, processing in progress, window complete, all channels complete.

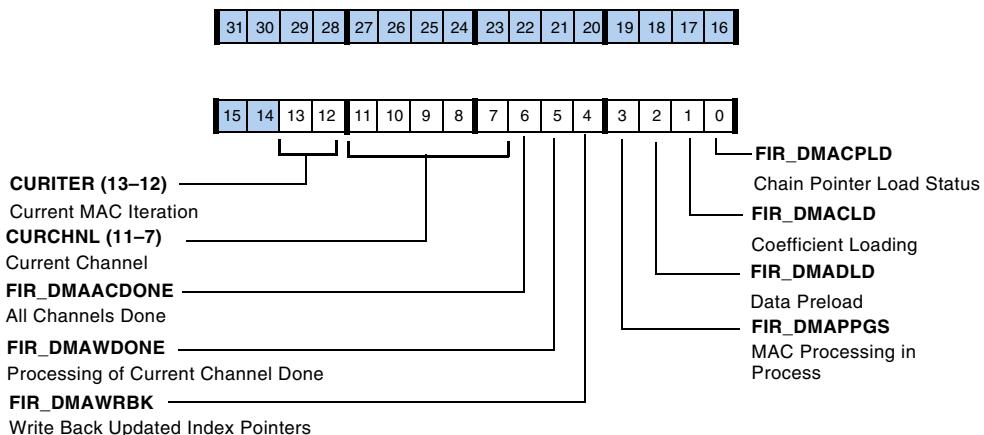


Figure A-38. FIRDMASTAT Register

Table A-50. FIRDMASTAT Register Bit Descriptions (RO)

Bits	Name	Description
0	FIR_DMACPLD	Chain Pointer Loading Status. High indicates state machine in chain pointer load state.
1	FIR_DMACLD	Coefficient Loading.
2	FIR_DMADLD	Data Preload.
3	FIR_DMAPPGS	MAC Processing in Progress.
4	FIR_DMARBK	Writing Back the Updated Index Registers.
5 (ROC)	FIR_DMAWDONE	Processing of Current Channel Done. (Sticky – Cleared on register read).
6 (ROC)	FIR_DMAACDONE	All Channels Done. (Sticky – Cleared on register read)

Peripheral Registers

Table A-50. FIRDMASTAT Register Bit Descriptions (RO)

Bits	Name	Description
11–7	CURCHNL	Current Channel. Channel that is being processed in the TDM slot. Zero indicates the last slot.
13–12	CURITER	Current MAC Iteration. Current MAC iteration in multi iteration mode. Zero indicates the final iteration.
31–14	Reserved	

FIR Debug Registers (FIRDEBUGCTL, FIRDBGADDR)

This register, shown in [Figure A-39](#) and described in [Table A-51](#), control the debug operation of the FIR accelerator and should only be used in debug mode.

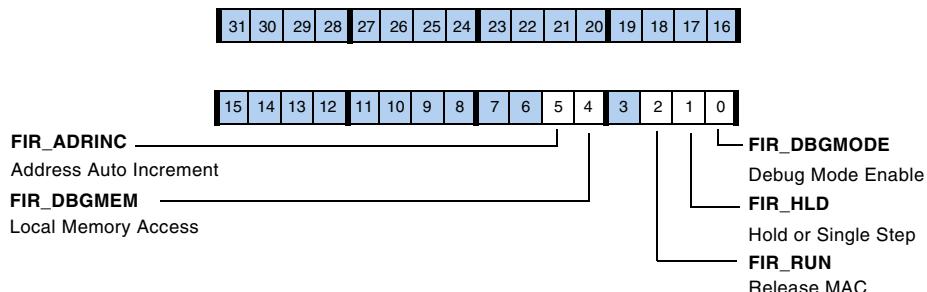


Figure A-39. FIRDEBUGCTL Register

Table A-51. FIRDEBUGCTL Register Bit Descriptions (RW)

Bits	Name	
0	FIR_DBGMODE	Debug Mode Enable. 0 = Disable 1 = Enable For local memory access, the FIRCTL1 register can be cleared.
1	FIR_HLD	Hold Or Single Step. The function of this bit is based on the FIR_DBGMEM bit setting. For FIR_DBGMEM = 0: 1 = Single step for FIR_DBGMEM = 1: 1 = Hold data
2 (WO)	FIR_RUN	Release MAC. This bit is self clearing after one FIR clock cycle.
3	Reserved	
4	FIR_DBGMEM	Local Memory Access. If set, the data and coefficients memory can be indirectly accessed.
5	FIR_ADRINC	Address Auto Increment. If this bit is set, the address register auto increments on DBGMEMWRDAT write and DBGMEMRDDAT reads.
31–6	Reserved	

IIR Accelerator Registers

The following sections describe the registers used to program and debug the FIR accelerator.

IIR Global Control Register (IIRCTL1)

The IIRCTL1 register, shown in [Figure A-40](#) and described in [Table A-52](#), is used to configure the global parameters for the accelerator. These include number of channels, channel auto iterate, DMA enable, and accelerator enable.

Peripheral Registers

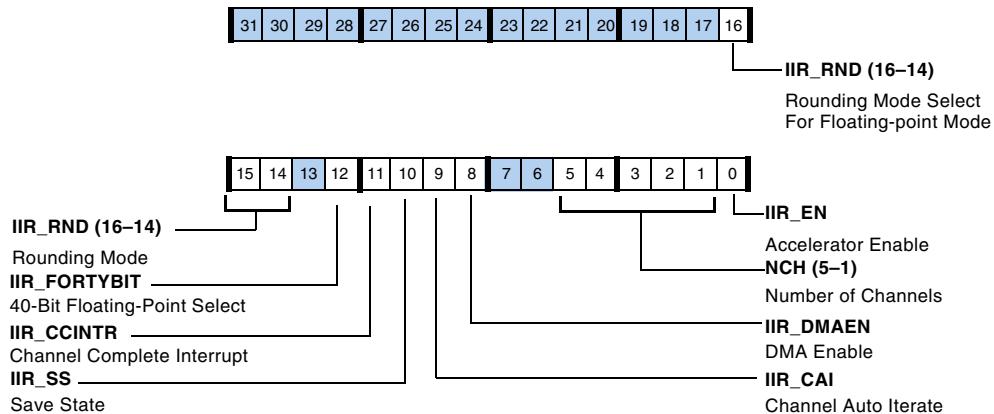


Figure A-40. IIRCTL1 Register

Table A-52. IIRCTL1 Register Bit Descriptions (RW)

Bits	Name	Description
0	IIR_EN	IIR Enable. 0 = IIR disabled 1 = IIR enabled
5–1	IIR_NCH	Number of Channels. Programmable between 0–23 Channels = NCH + 1
7–6	Reserved	
8	IIR_DMAEN	DMA Enable. 0 = Disable 1 = Enable
9	IIR_CAI	Channel Auto Iterate. 0 = Processing stops once all channels are over 1 = Moves to first channel and continues processing in a loop when all channels are over
10	IIR_SS	Save State. Stores the Dk registers settings into local memory.

Table A-52. IIRCTL1 Register Bit Descriptions (RW) (Cont'd)

Bits	Name	Description
11	IIR_CCINTR	Channel Complete Interrupt. 0 = Interrupt is generated only when all channels are done (default) 1 = Interrupt is generated after each channels is done (default)
12	IIR_FORTYBIT	40-Bit Floating-Point Format Select. 0 = 32-bit IEEE floating-point 1 = 40-bit IEEE floating-point
13	Reserved	
16–14	IIR_RND	Rounding Mode Select for Floating-Point Mode. 000 = IEEE round to nearest (even) 001 = IEEE round to zero 010 = IEEE round to +ve infinity 011 = IEEE round to -ve infinity 100 = Round to nearest Up 101 = Round away from zero 110 = Reserved 111 = Reserved
31–17	Reserved	

Peripheral Registers

IIR Channel Control Register (IIRCTL2)

The IIRCTL2 register, shown in [Figure A-41](#) and described in [Table A-53](#), is used to configure the channel specific parameters. These include number of biquads and window size.

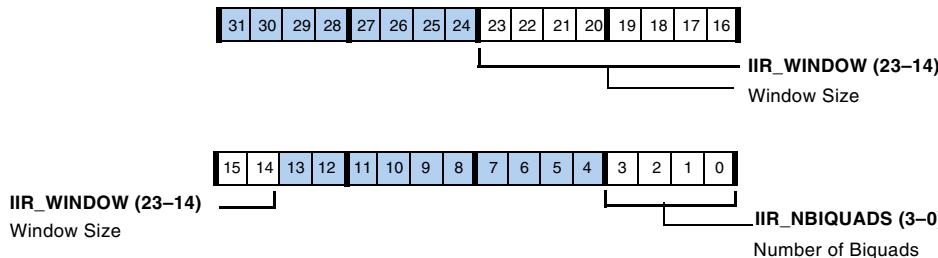


Figure A-41. IIRCTL2 Register

Table A-53. IIRCTL2 Register Bit Descriptions (RW)

Bits	Name	Description
3–0	IIR_NBIQUADS	Number of Biquads. Programmable between 0–11. Number of Biquads = BIQUADS + 1
13–4	Reserved	
23–14	IIR_WINDOW	Window Size Parameter. Programmable between 0–1023. Window Size = WINDOW+ 1
31–24	Reserved	

IIR MAC Status Register (IIRMACSTAT)

The IIRMACSTAT register, shown in [Figure A-42](#) and described in [Table A-54](#), provides the status of MAC operations.

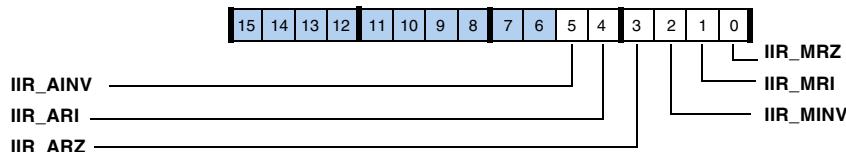


Figure A-42. IIRMACSTAT Register

Table A-54. IIRMACSTAT Register Bit Descriptions (RO)

Bits	Name	Description
0	IIR_MRZ	Multiplier Result Zero. Set if multiplier results is zero.
1	IIR_MRI	Multiplier Result Infinity. Set if multiplier results is infinity.
2	IIR_MINV	Multiply Invalid. Set if multiply operation is invalid.
3	IIR_ARZ	Adder Result Zero. Set if adder results is zero.
4	IIR_ARI	Adder Result Infinity. Set if adder results is infinity.
5	IIR_AINV	Addition Invalid. Set if addition is invalid.
31–6	Reserved	

IIR DMA Status Register (IIRDMASTAT)

The IIR DMA registers are described in “[Data Transfer](#)” on page 6-15. The IIRDMASTAT register, shown in [Figure A-43](#) and described in [Table A-55](#), provides the status of DMA operations. All the bits in this register are read only.

Peripheral Registers

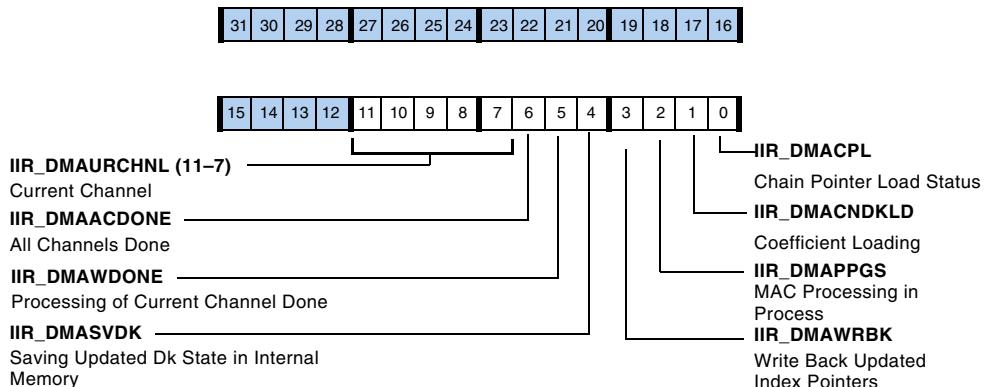


Figure A-43. IIRDMASTAT Register

Table A-55. IIRDMASTAT Register Bit Descriptions (RO)

Bits	Name	Description
0	IIR_DMACPL	Chain Pointer Loading Status. 1 = state machine in chain pointer load state
1	IIR_DMACnDkLD	Coefficient and Dk Loading.
2	IIR_DMAPPGS	MAC Processing In Progress.
3	IIR_DMAWRBK	Writing Back Updated Index Registers.
4	IIR_DMASVDK	Saving Updated Dk State in Internal Memory.
5 (ROC)	IIR_DMAWDONE	Processing of Current Channel Done. Sticky, cleared on register read.
6 (ROC)	IIR_DMAACDONE	All Channels Done. Sticky, cleared on register read.
11–7	IIR_DMAURCHNL	Current Channel. Channel that is being processed in the TDM slot. Zero indicates the last slot.
31–12	Reserved	

IIR Debug Registers (IIRDEBUGCTL, IIRDEBUGADDR)

The IIRDEBUGCTL register, shown in [Figure A-44](#) and described in [Table A-56](#), controls the debug mode operation of the IIR accelerator. Note that these registers should only be used in debug mode.

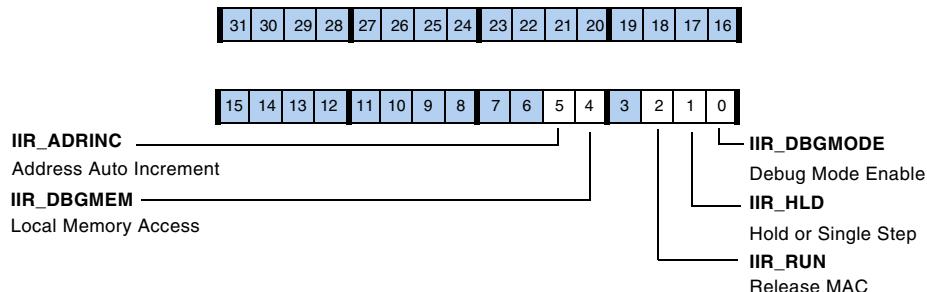


Figure A-44. IIRDEBUGCTL Register

Table A-56. IIRDEBUGCTL Register Bit Descriptions (RW)

Bits	Name	Description
0	IIR_DBGMODE	Debug Mode Enable. 0 = Disable 1 = Enable For local memory access, the IIRCTL1 register can be cleared.
1	IIR_HLD	Hold or Single Step. The function of this bit is based on the IIR_DEBUGMEM bit setting. For IIR_DEBUGMEM = 0: 1 = Single step For IIR_DEBUGMEM = 1: 1 = Hold data
2	IIR_RUN	Release the MAC. This bit is self clearing after one IIR clock cycle.
3	Reserved	
4	IIR_DEBUGMEM	Local Memory Access. If set, the data and coefficients memory can be indirectly accessed.

Peripheral Registers

Table A-56. IIRDEBUGCTL Register Bit Descriptions (RW) (Cont'd)

Bits	Name	Description
5	IIR_ADRINC	Address Auto Increment. If this bit is set, the address register auto increments on IIRDBGWRDATA_H/IIRDBGWRDATA_L writes and IIRDBGRDDATA_H/IIRDBGRDDATA_L reads.
31–6	Reserved	

Media Local Bus Registers

The registers in this section are used to program and get status information for the MLB interface and the specific channels used.

MLB Global Registers

This section lists all different control and status registers related to the MLB controller.

Device Control Configuration Register (MLB_DCCR)

This register, shown in [Figure A-45](#) and described in [Table A-57](#), controls the device enable/disable, clock rate, lock status and addressing.

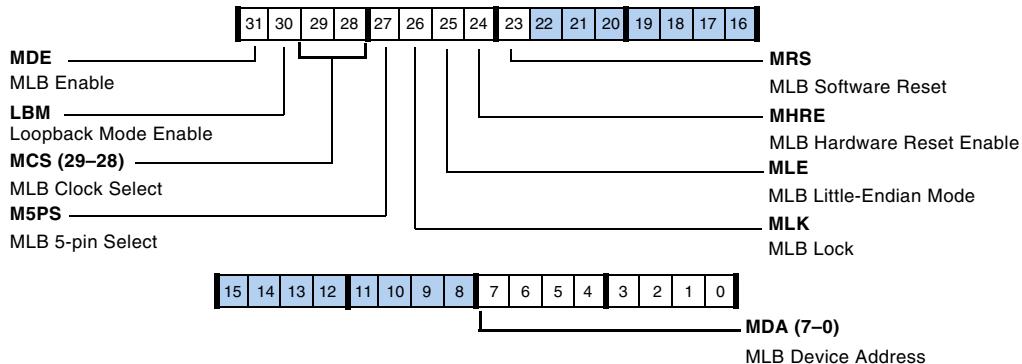


Figure A-45. MLB_DCCR Register

Table A-57. MLB_DCCR Register Bit Descriptions (RW)

Bit	Name	Description												
7–0	MDA	<p>MLB Device Address. Determines the unique device address (DA) for ADSP-214xx MediaLB device. MLB device address is 16 bits. Bits 15–9 and LSB are always zero. Only bits 8–1 vary and they are defined by MLB_DCCR bits 7–0. Device addresses are used by the system channel MLBSCAN command.</p> <table> <tr> <td>MDA</td> <td>Device Address</td> </tr> <tr> <td>00000001</td> <td>0000 000'0 0000 001'0 = 0x0002</td> </tr> <tr> <td>00000010</td> <td>0000 000'0 0000 010'0 = 0x0004</td> </tr> <tr> <td>00000011</td> <td>0000 000'0 0000 100'0 = 0x0006</td> </tr> <tr> <td>----</td> <td></td> </tr> <tr> <td>11111110</td> <td>0000 000'1 1111 110'0 = 0x01FC</td> </tr> </table> <p>For further information on assigning the device address, refer to the MLB specification.</p>	MDA	Device Address	00000001	0000 000'0 0000 001'0 = 0x0002	00000010	0000 000'0 0000 010'0 = 0x0004	00000011	0000 000'0 0000 100'0 = 0x0006	----		11111110	0000 000'1 1111 110'0 = 0x01FC
MDA	Device Address													
00000001	0000 000'0 0000 001'0 = 0x0002													
00000010	0000 000'0 0000 010'0 = 0x0004													
00000011	0000 000'0 0000 100'0 = 0x0006													

11111110	0000 000'1 1111 110'0 = 0x01FC													
22–8	Reserved													
23 (WO)	MRS	MLB Software Reset. When set, resets the MLB physical and link layer logic. Hardware clears this bit automatically.												
24	MHRE	<p>MLB Hardware Reset Enable. Enables hardware to automatically reset the MLB physical and link layer logic upon the reception MLB reset system command. 0 = Hardware reset option disabled 1 = MLB reset causes hardware reset</p>												
25	MLE	<p>MLB Little-Endian Mode. 0 = Big-Endian mode 1 = Little-Endian mode</p>												
26	MLK	<p>MLB Lock. When set, indicates that MLB port is synchronized to the incoming MLB frame. If MLK is clear (unlocked), it is set after FRAMESYNC is detected at the same position for three consecutive frames. If MLK is set (locked), it is cleared after not receiving FRAMESYNC at the expected time for two consecutive frames. While MLK is set, FRAME SYNC patterns occurring at locations other than the expected one are ignored.</p>												
27	M5PS	<p>MLB 5-Pin Select. 0 = 3-pin MLB mode 1 = 5-pin MLB mode</p>												

Peripheral Registers

Table A-57. MLB_DCCR Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
29–28	MCS	MLB Clock Select. 00 = 256Fs – supports 8 quadlets per frame 01 = 512Fs – supports 16 quadlets per frame 10 = 1024Fs – supports 32 quadlets per frame 11 = reserved
30	LBM	Loopback Mode Enable. 0 = Loopback disabled 1 = Loopback enabled
31	MDE	MLB Enable. 0 = MLB disabled 1 = MLB enabled

System Status Register (MLB_SSCR)

This register, shown in [Figure A-46](#) and described in [Table A-58](#), allows system software to monitor and control the status of the MLB network. The register is updated once per frame by hardware during the MLB system channel. The bits of this register are not valid until the ADSP-214xx is locked to the MLB interface (except for the bits associated with MLB lock and unlock, SDMU and SDML). System software must service events before the start of the next MLB frame to prevent the current frame status from being lost.

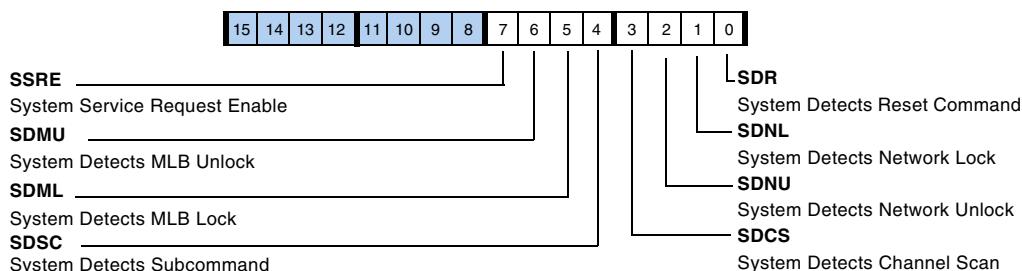


Figure A-46. MLB_SSCR Register

Table A-58. MLB_SSCR Register Bit Descriptions (RW1C)

Bit	Name	Description
0	SDR	System Detects Reset Command. When set, indicates MLB device has received MLBReset system command.
1	SDNL	System Detects Network Lock. When set, indicates the MLB device has received Most lock system command.
2	SDNU	System Detects Network Unlock. When set, indicates the MLB device has received MOST unlock system command.
3	SDCS	System Detects Channel Scan. When set, indicates the MLB device has received the MLBScan system command. The device address is stored in the SDCR register.
4	SDSC	System Detects Subcommand. When set, indicates the MLB device has received the MLBSubCmd system command. The device command is stored in the SDCR register and decoding is performed by software.
5	SDML	System Detects MLB Lock. When set, indicates the MLB device has locked onto the MLB frame.
6	SDMU	System Detects MLB Unlock. When set, indicates the MLB device has unlocked from the MLB frame.
7 (RW)	SSRE	System Service Request Enable. When set, indicates that the MLB device needs service. Write 1 to set.
31–8	Reserved	

System Data Configuration Register (MLB_SDCR)

This register, described in [Table A-59](#), allows system software to receive control information from the MLB controller. The `MLB_SDCR` register is updated once per frame by the hardware during the MLB system channel. This register is loaded with the data from the `MLBDAT_IN` signal during the system channel quadlet. System software must read the `MLB_SDCR` register before the start of the next MLB frame to prevent the current data from being lost.

Peripheral Registers

Table A-59. MLB_SDCR Register Description (RO)

Bit	Name	Description
31–0	SDATA	System Channel Data.

System Mask Configuration Register (MLB_SMCR)

This register, described in [Table A-60](#), allows system software to mask system status interrupts. When a mask bit is set, the corresponding system channel interrupt is masked.

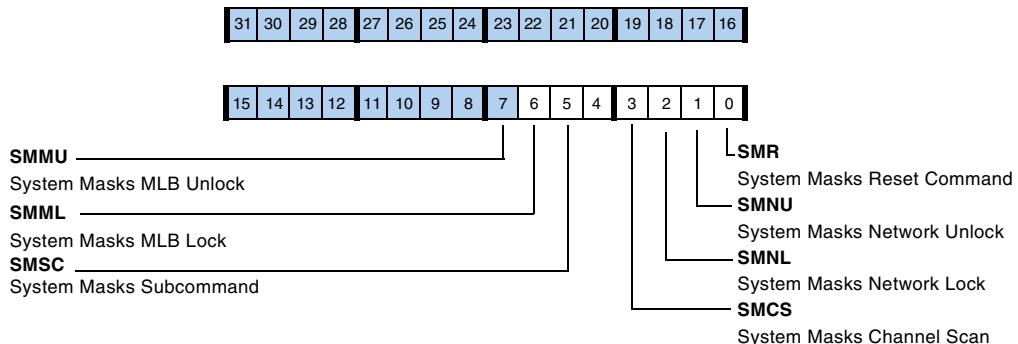


Figure A-47. MLB_SMCR Register

Table A-60. MLB_SMCR Register Bit Descriptions (RW)

Bit	Name	Description
0	SMR	System Masks Reset Command. When set, this bit masks system interrupts for Mlb Reset system command.
2	SMNU	System Masks Network Unlock. When set, this bit masks system interrupts for the MOST_unlock system command.
1	SMNL	System Masks Network Lock. When set, this bit masks system interrupts for the MOST_Lock system command.
3	SMCS	System Masks Channel Scan. When set, this bit masks system interrupts for MlbScan system command.

Table A-60. MLB_SMCR Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
4	SMSC	System Masks Subcommand. When set, this bit masks system interrupts for MlbSubCmd (0xE6) system command.
5	SMML	System Masks MLB Lock. When set, this bit masks system interrupts generated when MLB lock is detected. At reset, MLB lock events are masked, (SMML = 1).
6	SMMU	System Masks MLB Unlock. When set, this bit masks system interrupts generated when a MediaLB unlock is detected. At reset, MediaLB unlock events are masked (SMMU = 1).
31–7	Reserved	

Channel Interrupt Status Register (MLB_CICR)

The channel interrupt status register reflects the channel interrupt status of the individual logical channels. The channel status update (CSU) bits are set by hardware when a channel interrupt is generated. The CSU bits are sticky and can only be reset by software. To clear a particular bit in this register, software must clear all of the unmasked status bits in the corresponding `MLB_CSCRx` registers.

Table A-61. MLB_CICR Register Description (RO)

Bit	Name	Description
30–0	CSU	Channel Status Update.
31	Reserved	

MLB Base Registers

The DMA address is constituted by a 5-bit base in the MLB base registers (for the corresponding channel data type) and a 14-bit offset configured using the BCA bits in the `MLB_CCBCRx` register. The base address registers and offset registers use round robin arbitration to determine which logical channel is granted access to the DMA bus.

Peripheral Registers

Synchronous Base Address Register (MLB_SBCR)

The `MLB_SBCR`, described in [Table A-62](#), holds the base address of the system memory buffers of all synchronous channels in the device.

Table A-62. `MLB_SBCR` Register Bit Descriptions (RW)

Bit	Name	Description
4–0	STBA	Synchronous transmit base address for DMA mode
15–5	Reserved	
20–16	SRBA	Synchronous receive base address for DMA mode
31–21	Reserved	

Asynchronous Base Address Register (MLB_ABCR)

The `MLB_ABCR` register, described in [Table A-63](#), holds the base address of the system memory buffers of all asynchronous channels in the device.

Table A-63. `MLB_ABCR` Register Bit Descriptions (RW)

Bit	Name	Description
4–0	ATBA	Asynchronous transmit base address for DMA mode
15–5	Reserved	
20–16	ARBA	Asynchronous receive base address for DMA mode
31–21	Reserved	

Control Base Address Register (MLB_CBCR)

The `MLB_CBCR` register, described in [Table A-64](#), hold the base address of the system memory buffers of all control channels in the device.

Table A-64. `MLB_CBCR` Register Bit Descriptions (RW)

Bit	Name	Description
4–0	CTBA	Control transmit base address for DMA mode
15–5	Reserved	
20–16	CRBA	Control receive base address for DMA mode
31–21	Reserved	

Isynchronous Base Address Register (MLB_IBCR)

The `MLB_IBCR` register, described in [Table A-65](#), holds the base address of the system memory buffers of all isynchronous channels in the device.

Table A-65. `MLB_ABCR` Register Bit Descriptions (RW)

Bit	Name	Description
4–0	ATBA	Isynchronous transmit base address for DMA mode
15–5	Reserved	
20–16	ARBA	Isynchronous receive base address for DMA mode
31–21	Reserved	

Logical Channel Registers

The MLB controller supports up to 31 logical channels. Therefore the variable in the register names is valid for $x = 0\text{--}30$. This section lists all different control and status registers related to the logical channels.

Peripheral Registers

Channel Control Registers (MLB_CECRx)

These registers define the basic attributes of a given logical channel, such as the channel enable, channel direction, and channel address. The definition of the bit fields in these registers vary by the selected channel type.

[Figure A-48](#) and [Table A-66](#) provide information for for asynchronous and control channels and [Figure A-49](#) and [Table A-67](#) provide information for for synchronous channels.

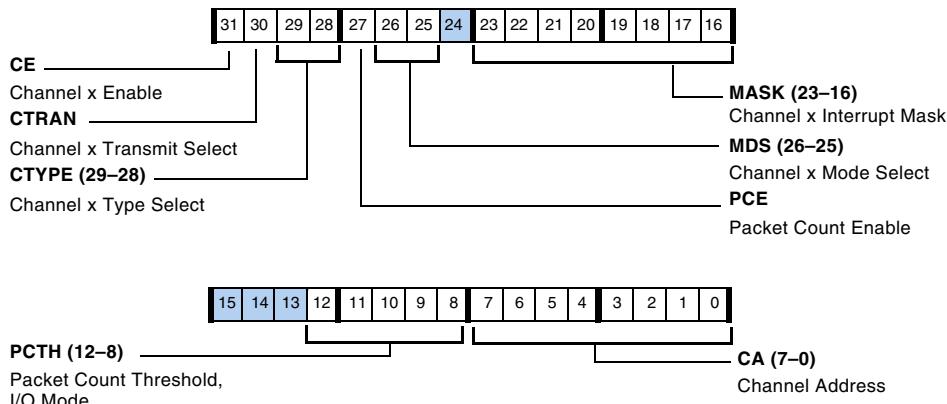


Figure A-48. MLB_CECRx Register (Asynch and Control Channels)

Table A-66. MLB_CECRx Register Bit Descriptions for Asynchronous and Control Channels (RW)

Bit	Name	Description												
7–0	CA	<p>Channel Address. These bits determine the channel address associated with this logical channel. MLB channel address is 16 bits; bits 15–9 and LSB are always zero. Only bits 8–1 vary and they are defined by MLB_CECRx bits 7–0.</p> <p>Channel Address</p> <table> <tr><td>00000001</td><td>0x0002</td></tr> <tr><td>00000010</td><td>0x0004</td></tr> <tr><td>00000011</td><td>0x0006</td></tr> <tr><td>00000100</td><td>0x0008</td></tr> <tr><td>.....</td><td></td></tr> <tr><td>11111111</td><td>0x01FE</td></tr> </table> <p>For further information on assigning the device address, refer to the MLB specification.</p>	00000001	0x0002	00000010	0x0004	00000011	0x0006	00000100	0x0008		11111111	0x01FE
00000001	0x0002													
00000010	0x0004													
00000011	0x0006													
00000100	0x0008													
.....														
11111111	0x01FE													
12–8	PCTH	Packet Count Threshold, I/O Mode. Software can program this field with the number of packets to receive before generating an Rx packet-count service request. This service request is generated independent of, and in addition to, other service requests generated via the standard buffer threshold mechanism. In DMA mode these bits are reserved.												
15–13	Reserved													
16	MASK0	Mask Protocol Error. When set, masks protocol error channel interrupts for this logical channel. This bit valid for all Rx channel types. This is valid for asynchronous and control Tx channels only.												
17	MASK1	Mask Detect Break. When set, masks detect break channel interrupt for this logical channel. This bit is valid for asynchronous and control channels only.												
18	MASK2 (I/O)	Masks Receive Service Request. When set, masks Rx channel service request interrupts for this logical channel.												
18	MASK2 (DMA)	Mask Buffer Done. When set, masks buffer done channel interrupts for this logical channel.												
19	MASK3 (I/O)	Masks Transmit Service Request. When set, masks Tx channel service request interrupts for this logical channel.												
19	MASK3 (DMA)	Mask Buffer Start. When set, masks buffer start channel interrupts for this logical channel.												

Peripheral Registers

Table A-66. MLB_CECRx Register Bit Descriptions for Asynchronous and Control Channels (RW) (Cont'd)

Bit	Name	Description
20	MASK4	Mask Buffer Error. When set, masks buffer error channel interrupts for this logical channel.
21	Reserved	
22	MASK6	Mask Lost Frame Synchronization. When set, masks lost frame synchronization channel interrupts for this logical channel.
23	MASK7	Reserved
24	Reserved	
26–25	MDS	Channel x Mode Select. 00 = Ping-pong DMA mode (default) 01 = Circular buffering DMA 10 = I/O mode enable 11 = Reserved
27	PCE	Packet Count Enable. Enable the Rx packet counter. This bit is valid for asynchronous and control Rx channels in I/O mode. 0 = Disable 1 = Enable
29–28	CTYPE	Channel x Type Select. 00 = Synchronous (default) 01 = Reserved 10 = Asynchronous 11 = Control
30	CTRAN	Channel x Transmit Select. 0 = Receive (default) 1 = Transmit
31	CE	Channel x Enable. 0 = Channel n disabled (default) 1 = Enabled

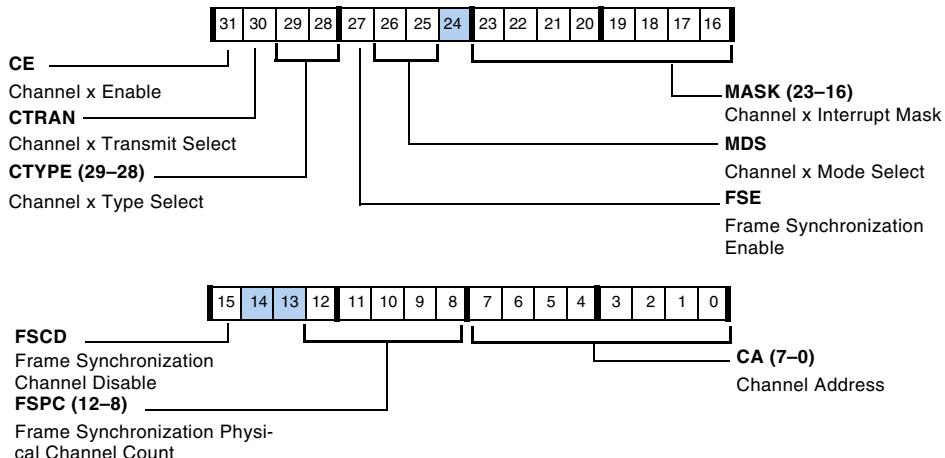


Figure A-49. MLB_CECRx Register (Synchronous Channels)

Table A-67. MLB_CECRx Register Bit Descriptions for Synchronous Channels (RW)

Bit	Name	Description
7–0	CA	Channel Address. These bits determine the channel address associated with this logical channel. MLB channel address is 16 bits; bits 15–9 and LSB are always zero. Only bits 8–1 vary and they are defined by MLB_CECRx bits 7–0. 00000001 0x0002 00000010 0x0004 00000011 0x0006 00000100 0x0008 11111111 0x01FE
12–8	FSPC	Frame Synchronization Physical Channel Count. Defines the number of physical channels expected to match this logical channel's channel address each MLB frame.
14–13	Reserved	
15	FSCD	Frame Synchronization Channel Disable. When set, disables this logical channel when frame synchronization is lost.

Peripheral Registers

Table A-67. MLB_CECRx Register Bit Descriptions for Synchronous Channels (RW)

Bit	Name	Description
16	MASK0	Mask Protocol Error. When set, masks protocol error channel interrupts for this logical channel. This bit valid for all Rx channel types. This is valid for asynchronous and control Tx channels only.
17	MASK1	Mask Detect Break. When set, masks detect break channel interrupt for this logical channel. This bit is valid for asynchronous and control channels only.
18	MASK2 (I/O)	Masks Receive Service Request. When set, masks Rx channel service request interrupts for this logical channel.
18	MASK2 (DMA)	Mask Buffer Done. When set, masks buffer done channel interrupts for this logical channel.
19	MASK3 (I/O)	Masks Transmit Service Request. When set, masks Tx channel service request interrupts for this logical channel.
19	MASK3 (DMA)	Mask Buffer Start. When set, masks buffer start channel interrupts for this logical channel.
20	MASK4	Mask Buffer Error. When set, masks buffer error channel interrupts for this logical channel.
21	MASK5	Reserved
22	MASK6	Mask Lost Frame Synchronization. When set, masks lost frame synchronization channel interrupts for this logical channel.
23	MASK7	Reserved
24	Reserved	
26–25	MDS	Channel x Mode Select. 00 = Ping-pong DMA mode (default) 01 = Circular buffering DMA 10 = I/O mode enable 11 = Reserved
27	FSE	Frame Synchronization Enable. When set, enables streaming channel frame synchronization for this logical synchronous channel.

Table A-67. MLB_CECRx Register Bit Descriptions for Synchronous Channels (RW)

Bit	Name	Description
29–28	CTYPE	Channel x Type Select. 00 = Synchronous (default) 01 = Reserved 10 = Asynchronous 11 = Control
30	CTRAN	Channel x Transmit Select. 0 = Receive (default) 1 = Transmit
31	CE	Channel x Enable. 0 = Channel x disabled (default) 1 = Enabled

Peripheral Registers

Channel Status Configuration Registers (MLB_CSCRx)

This register, shown in [Figure A-50](#) and described in [Table A-68](#), shows the status of the current and previous buffer for the given logical channel. For all bits a 1 means the condition exists.

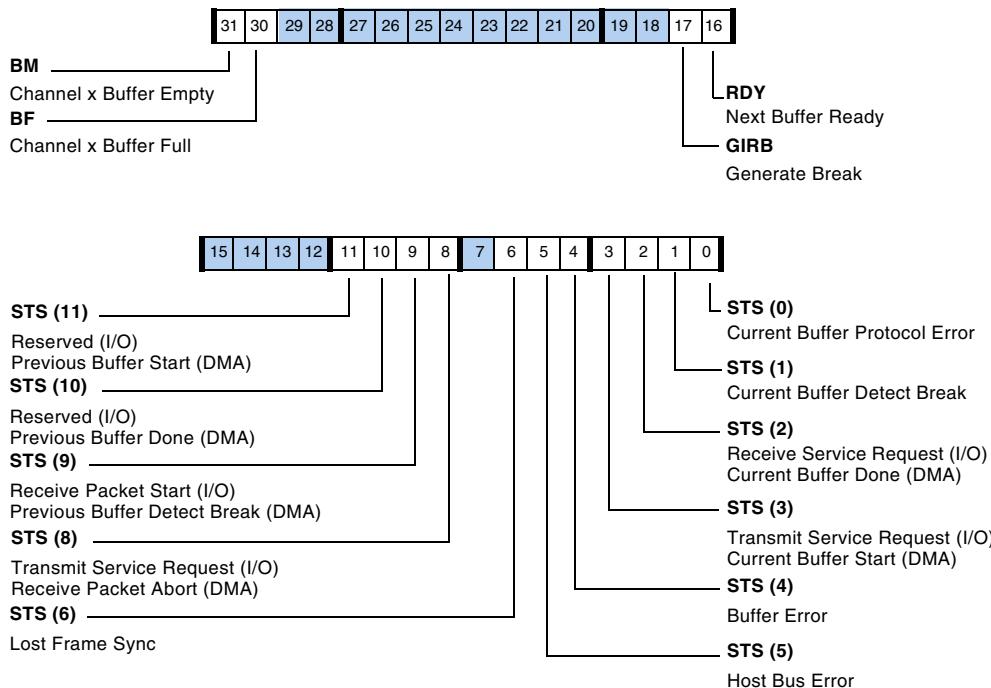


Figure A-50. MLB_CECRx Register

Table A-68. MLB_CSCRx Register Description (RO)

Bit	Name	Description
0	STS0	Current Buffer Protocol Error. Indicates that either a TX channel has detected an RxStatus of ReceiverProtocolError (72h), an RX channel has detected an invalid command for a given channel type, or an additional ControlStart (30h) or AsyncStart (20h) command has been received while in the middle of a packet. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all RX channel types and valid for only asynchronous and control TX channels.
1	STS1	Current Buffer Detect Break. Indicates that either a TX channel has detected a receiver break response, ReceiverBreak (70h), or an RX channel has detected a transmitter break command, ControlBreak (36h) or AsyncBreak (26h), while processing the Current Buffer. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for asynchronous and control channels only.
2	STS2 (I/O)	Receive Service Request (I/O). Indicates that an RX channel is requesting service from system software. Receive service requests are issued if the number of free quadlets in the local channel buffer is less than or equal to LCBCRn.TH[9:0]. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types.
2	STS2 (DMA)	Current Buffer Done (DMA). Indicates that the last quadlet from the last packet (in the Current Buffer) has been successfully transmitted or received. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types.
3	STS3 (I/O)	Transmit Service Request. Indicates that a TX channel is requesting service from system software. Transmit service requests are issued if the number of valid quadlets in the local channel buffer is less than or equal to LCBCRn.TH[9:0]. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types.
3	STS3 (DMA)	Current Buffer Start. Indicates that the DMA controller has started processing the Current Buffer. This bit is set after the contents of CNBCRn have been loaded into CCBCRn, the CSCRn.RDY bit has been cleared (for ping-pong buffering), and hardware is available to accept the next buffer. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types.

Peripheral Registers

Table A-68. MLB_CSCRx Register Description (RO) (Cont'd)

Bit	Name	Description
4	STS4	Buffer Error. Indicates that either a TX channel has detected a buffer underflow (attempt to pop data from an empty buffer), or an RX channel has detected a buffer overflow (attempt to push data onto a full buffer). The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for synchronous RX/TX (CECRn.FCE = 0) channels only.
5	STS5 (DMA)	Host Bus Error. Indicates that an HBI bus error has been detected. The setting of this bit generates a non-maskable channel interrupt to system software.
6	STS6	Lost Frame Sync. Indicates that the logical channel has lost synchronization with the MediaLB frame. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for synchronous channels only.
7	Reserved	
8	STS8 (I/O)	Receive Packet Abort. Indicates that an RX channel has detected an aborted packet. Received packets are aborted if the receiver generates a break response, ReceiverBreak (70h), or detects a transmitter packet break command; ControlBreak (36h) or AsyncBreak (26h). This bit can also indicate the RX channel has detected a transmit command protocol error. The setting of this bit generates a maskable channel interrupt to system software. This interrupt can be used by system software to detect when it has encountered the beginning of an aborted packet. This bit is valid for asynchronous and control RX channels only
8	STS8 (DMA)	Previous Buffer Protocol Error. Indicates that either a TX channel has detected an RxStatus of ReceiverProtocolError (72h), a RX channel has detected an invalid command for this channel type, or an additional AsyncStart (20h) or ControlStart (30h) command has been received while in the middle of a packet. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all RX channels and valid for only asynchronous and control TX channels.
9	STS9 (I/O)	Receive Packet Start. Indicates that an RX channel has detected a transmitter packet start command; ControlStart (30h) or AsyncStart (20h). This status bit can be used by system software to detect when it has reached the end of an aborted packet. This bit is valid for asynchronous and control RX channels only.
9	STS9 (DMA)	Previous Buffer Detect Break. Indicates that either a TX channel has detected a receiver break response, ReceiverBreak (70h), or an RX channel has detected a transmitter break command, ControlBreak (36h) or AsyncBreak (26h), while processing the Previous Buffer. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types.

Table A-68. MLB_CSCRx Register Description (RO) (Cont'd)

Bit	Name	Description
10	STS10 (DMA)	Previous Buffer Done. Indicates that the last quadlet of the Previous Buffer has been successfully transmitted or received. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types. Reserved in I/O mode.
11	STS11 (DMA)	Previous Buffer Start. Indicates the first quadlet of the Previous Buffer has been successfully transmitted or received. The setting of this bit generates a maskable channel interrupt to system software. This bit is valid for all channel types. Reserved in I/O mode.
15–12	Reserved	
16 (RW)	RDY	Next Buffer Ready (DMA Mode). This bit is reserved for I/O mode. 0 = Next buffer ready for ping-pong DMA. Hardware clears this bit after the buffer begins to be processed. 1 = Next buffer ready for circular buffer DMA. Software should clear this bit only when buffer processing needs to be stopped.
17 (RW)	GIRB	Reserved = For Synchronous Channels. 0 = Generate isochronous receive break for isochronous channels. 1 = Generate break for asynchronous and control channels.
29–18	Reserved	
30	BF	Channel x Buffer Full. When set, buffer is full.
31	BM	Channel x Buffer Empty. When set, buffer is empty.

Channel x Current Buffer Configuration Registers (MLB_CCBCRx)

These read-only registers, described in [Table A-69](#), allow software to monitor the address pointer and buffer length of the current DMA buffer in internal memory when the logical channel is configured in DMA mode. When configured in I/O mode, this register implements the Rx data buffer. The definition of the bit fields in this register vary depending on the selected channel type.

Peripheral Registers

The 5-bit offset of the DMA address comes from the base address registers.

Table A-69. MLB_CCBCRx Register Bit Descriptions (RO)

Bit	Name	Description
1–0		Reserved for other channel types
15–2	BFA	Buffer Final Address.
17–16		Reserved for other channel types
31–18	BCA	Buffer Current Address.

Channel x Next Buffer Configuration Registers (MLB_CNBCRx)

These registers, described in [Table A-70](#), allows system software to set the start and end address of the next buffer in internal memory for the logical channel in DMA mode. When configured in I/O mode, these registers implement the transmit data buffer. The definition of bit fields in this register vary dependant on the selected channel type and are read-write.

Table A-70. MLB_CNBCRx Register Description (RW)

Bit	Name	Description
1–0		Reserved for other channel types
15–2	BEA	Next Buffer End Address.
17–16	Reserved	
31–18	BSA	Next Buffer Start Address.

Local Buffer Configuration Registers (MLB_LCBCRx)

These registers, described in [Table A-71](#), allow software to optimize the use of the local channel buffer memory. These registers should only be written by software while the logical channel is disabled. The size of the local channel buffer RAM is 124 words. At reset, this RAM is shared

equally by all 31 channels with four words for each channel. The buffer depth can be up to 124 words (quadlets), when only one channel is used.

Table A-71. MLB_LCBCRx Register Description (RW)

Bit	Name	Description
12–0	SA	Buffer Start Address. Determines the starting address (in quadlets/4) of the channel buffer for the logical channel x. Reset value = {120, 116, 112 ...4, 0} for x = 30:0 0x0000 = start address offset of 0 words 0x0001 = start address offset of 4 words 0x0002 = start address offset of 8 words ... 0x001E = start address offset of 120 words 0x001F to 0x1FFF = Reserved
21–13	BD	Buffer Depth. Defines the depth (in quadlets/4–1) of the local channel buffer for the logical channel x. 0x000 = depth = 4 words 0x001 = depth = 8 words 0x002 = depth = 12 words ... 0x01E = depth = 124 words 0x01F to 0x1FF = Reserved Reset value = 4 for all x=0:30
31–22	TH	Buffer Threshold. Defines the threshold (in quadlets/2) of the local channel buffer for the logical channel x in I/O mode. Hardware uses this threshold value to determine when to issue an I/O service request to system software. Reset value = 2 for all x = 0:30 0x000 = threshold = 0 word 0x001 = threshold = 2 words 0x002 = threshold = 4 words ... 0x03D = threshold = 122 words 0x03E to 0x3FF = Reserved Reserved in DMA mode

Peripheral Registers

Watchdog Timer Registers

The following sections provide bit descriptions for the registers associated with the watchdog timer.

Control (WDTCTL)

The watchdog control register (WDTCTL), is a 32-bit system memory-mapped register used to configure the watchdog timer. WDTCTL is protected against accidental writes from the processor core by the watchdog unlock register (WDTUNLOCK). Attempts by the core to write to WDTCTL without an unlock command causes the WDT to expire, and reset the system. This condition is captured in the watchdog exception field (WDERR).

Writes made by software to this register keep it enabled. Only an External hardware reset can disable WDTCTL.

Status (WDTSTATUS)

The WDTSTATUS register, shown in [Figure A-51](#) and described in [Table A-72](#), contains the watchdog timer status information. This register is not cleared by the WDT generated reset.

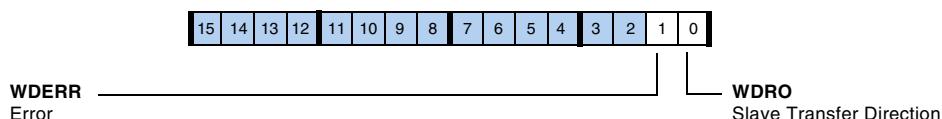


Figure A-51. WDTSTATUS Register

Table A-72. WDTSTATUS Register Bit Descriptions (RO)

Bit	Name	Description
0 (W1C)	WDRO	Watchdog Expired. Indicates that DSP core attempted to write to WDT configuration space without an unlock “command”. Bit is set when the above exception occurs. Software can determine whether the watchdog has expired by interrogating this bit. This is a sticky bit that is set whenever the watchdog timer count reaches 0.
1 (W1C)	WDERR	Watchdog Error. Indicates that watchdog timer has expired. Bit is set when counter expires. Attempts by the core to write to the WDT configuration space without an unlock command, causes the WDT to expire and this condition is captured in the watchdog exception field. This is a sticky bit that is set whenever the above exception occurs.

Current Count (WDTCURCNT)

The WDTCURCNT register contains the current count value of the watchdog timer. Reads to WDTCURCNT return the current count value. For added safety, this register can only be updated when WDT configuration space is unlocked by programming the command in the WDTUNLOCK register. Values cannot be stored directly in WDTCURCNT, but are instead copied from WDTCNT.

Enabling the watchdog timer does not automatically reload WDTCURCNT from WDTCNT. The WDTCURCNT register is a 32-bit unsigned system memory-mapped register that must be accessed with 32-bit reads and writes.

Trip Counter (WDTTRIP)

The WDT contains a software programmable register WDTTRIP that sets the number of times that the WDT can expire before the WDTRST0 pin is continually asserted until the next time hardware reset is applied. This register is unaffected by WDT generated reset. This register can only be

Peripheral Registers

updated when the WDT is disabled and WDT configuration space is unlocked by programming the command in the `WDTUNLOCK` register.

Table A-73. WDTTRIP Register Bit Descriptions (RW)

Bit	Name	Description
3–0	TRIPVAL	Current Value of Trip Counter. This is the trip counter value, programmable from 0 to 15. The number of times WDT can expire is programmable by the TRIPVAL field. Reading this register also gives the current value of the trip counter.
7–4 (RO)	CURTRIPVAL	Current Number of WDT Resets. Reports all current WDT generated resets (<code>WDTRSTO</code> asserted).

Clock Select (WDTCLKSEL)

This register, described in [Table A-74](#), can only be updated when the WDT is disabled and WDT configuration space is unlocked by programming the command in the `WDTUNLOCK` register. Writes to the `WDTCLKSEL` register are ignored after the WDT is enabled.

Note: This register is reset on external hardware reset only. This ensures that the selected clock source remains the same even after a WDT generated reset is asserted.

Table A-74. WDTCLKSEL Register Bit Descriptions (RW)

Bit	Name	Description
0	CLKSEL	Clock Select. When this bit = 0, the WDTCLK source can be an external clock applied to the <code>WDT_CLKIN</code> pin or an external ceramic Oscillator connected to the <code>WDT_CLKIN</code> and <code>WDT_CLKO</code> pins. 0 = Selects ceramic oscillator output or external clock 1 = Selects internal RC oscillator output

Table A-74. WDTCLKSEL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
1	OSCPWRDWN	Internal RC Oscillator Power Down. 0 = Oscillator is powered up 1 = Oscillator is powered down
2	OSCRST	Internal RC Oscillator Reset. 0 = Oscillator is reset 1 = Oscillator out of reset

Period (WDTCNT)

The `WDTCNT` register, shown in Table 3, holds the 32-bit unsigned count value. The `WDTCNT` register must always be accessed with 32-bit read/writes.

The watchdog count register holds the programmable count value. A valid write to the watchdog count register also preloads the watchdog current counter. For added safety, the watchdog count register can only be updated when the WDT is disabled and WDT configuration space is unlocked by programming the command in the `WDTUNLOCK` register.

Unlock (WDTUNLOCK)

The `WDTUNLOCK` register protects the WDT configuration space against accidental writes from the processor core. Before attempting to write to the WDT configuration space, the core must unlock the WDT by writing the command value (0xAD21AD21) to this register. Attempts by the core to write to WDT configuration space without this command causes the WDT to expire. This exception is captured in the `WDTSTATUS` register. After configuring the WDT configuration space, the core needs to lock it again by writing any value other than the command value to the `WDTUNLOCK` register.

DAI Signal Routing Unit Registers

The digital applications interface is comprised of a group of peripherals and the signal routing unit (SRU). These register groups are described in the sections that follow.

Clock Routing Control Registers (SRU_CLKx, Group A)

These registers (see [Figure A-52](#) through [Figure A-57](#)) correspond to the group A clock sources listed in [Table A-75](#). Each of the clock inputs are connected to a clock source, based on the 5-bit values in the figures. When either of the precision clock generators is used in external source mode, the SRU_CLK3 register, pins 0–4 and/or pins 5–9, specify the source.



SPORTs 6 and 7 receive their clocks from other routed sources but cannot route their own clocks to other SPORTs or other peripherals internally through the SRU. If externally needed, they have to be routed through the DAI pins.

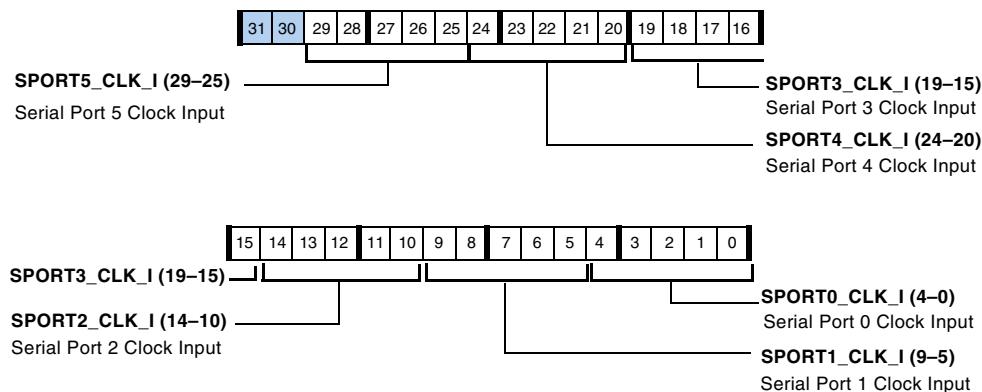


Figure A-52. SRU_CLK0 Register (RW)

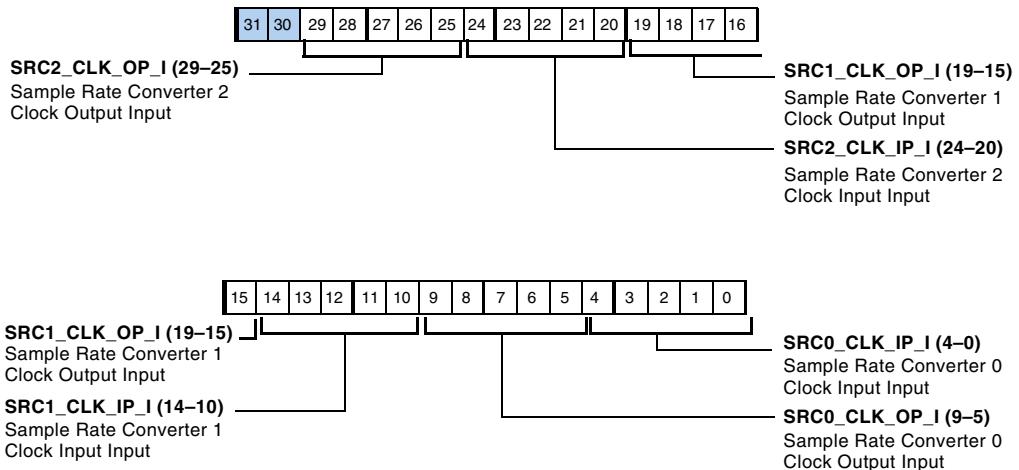


Figure A-53. SRU_CLK1 Register (RW)

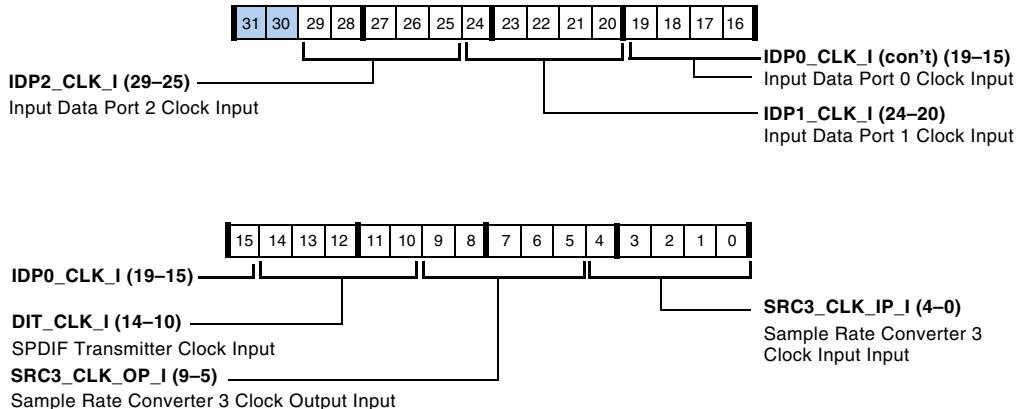


Figure A-54. SRU_CLK2 Register (RW)

DAI Signal Routing Unit Registers

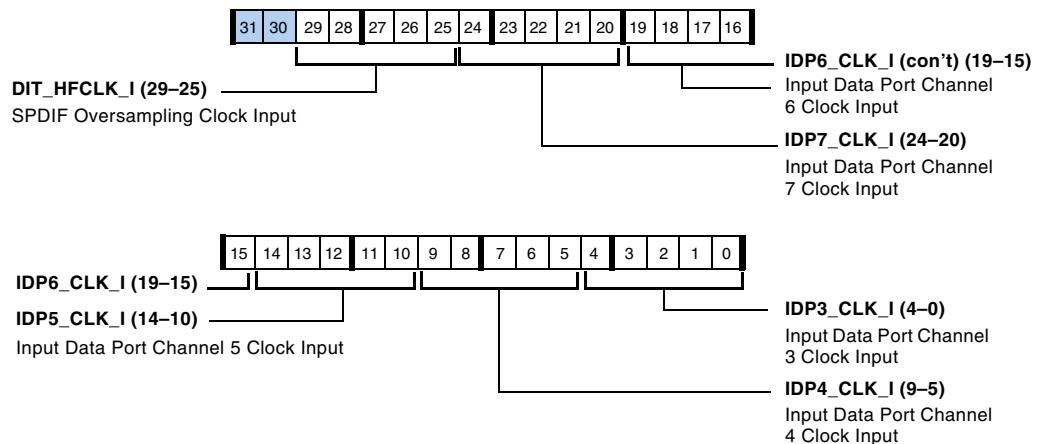


Figure A-55. SRU_CLK3 Register (RW)

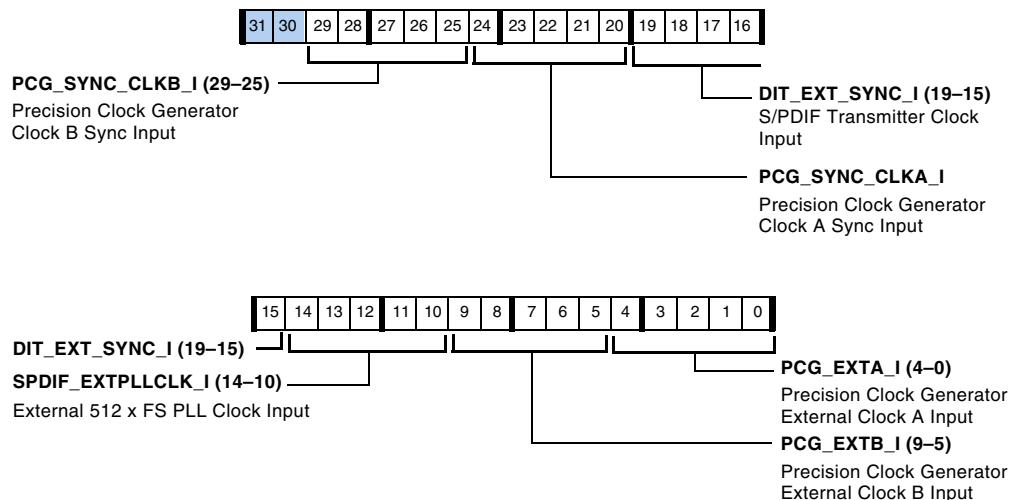


Figure A-56. SRU_CLK4 Register (RW)

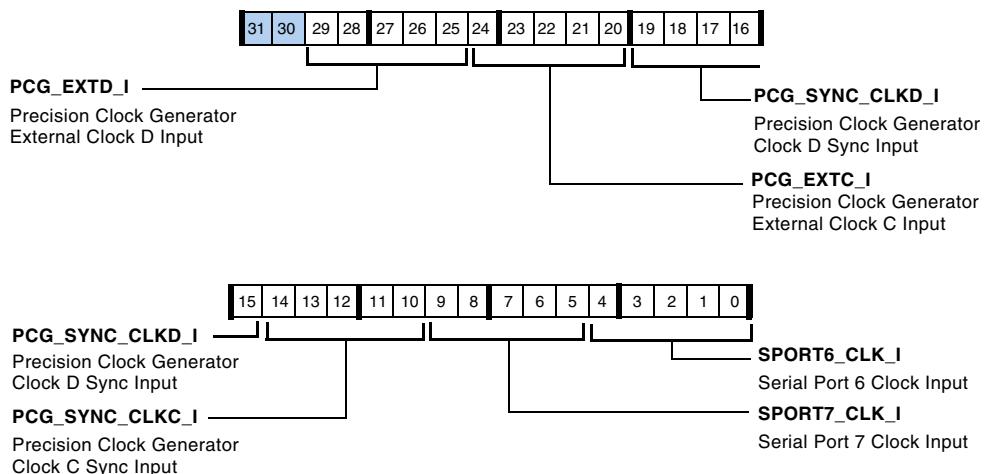


Figure A-57. SRU_CLK5 Register (RW)

Table A-75. Group A Sources – Serial Clock

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI_PB01_O	Pin Buffer 1
00001 (0x1)	DAI_PB02_O	Pin Buffer 2
00010 (0x2)	DAI_PB03_O	Pin Buffer 3
00011 (0x3)	DAI_PB04_O	Pin Buffer 4
00100 (0x4)	DAI_PB05_O	Pin Buffer 5
00101 (0x5)	DAI_PB06_O	Pin Buffer 6
00110 (0x6)	DAI_PB07_O	Pin Buffer 7
00111 (0x7)	DAI_PB08_O	Pin Buffer 8
01000 (0x8)	DAI_PB09_O	Pin Buffer 9
01001 (0x9)	DAI_PB10_O	Pin Buffer 10
01010 (0xA)	DAI_PB11_O	Pin Buffer 11
01011 (0xB)	DAI_PB12_O	Pin Buffer 12
01100 (0xC)	DAI_PB13_O	Pin Buffer 13

DAI Signal Routing Unit Registers

Table A-75. Group A Sources – Serial Clock (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
01101 (0xD)	DAI_PB14_O	Pin Buffer 14
01110 (0xE)	DAI_PB15_O	Pin Buffer 15
01111 (0xF)	DAI_PB16_O	Pin Buffer 16
10000 (0x10)	DAI_PB17_O	Pin Buffer 17
10001 (0x11)	DAI_PB18_O	Pin Buffer 18
10010 (0x12)	DAI_PB19_O	Pin Buffer 19
10011 (0x13)	DAI_PB20_O	Pin Buffer 20
10100 (0x14)	SPORT0_CLK_O	SPORT 0 Clock
10101 (0x15)	SPORT1_CLK_O	SPORT 1 Clock
10110 (0x16)	SPORT2_CLK_O	SPORT 2 Clock
10111 (0x17)	SPORT3_CLK_O	SPORT 3 Clock
11000 (0x18)	SPORT4_CLK_O	SPORT 4 Clock
11001 (0x19)	SPORT5_CLK_O	SPORT 5 Clock
11010 (0x1A)	DIR_CLK_O	SPDIF Receive Clock Output
11011 (0x1B)	DIR_TDMCLK_O	SPDIF Receive TDM Clock Output
11100 (0x1C)	PCG_CLKA_O	Precision Clock A Output
11101 (0x1D)	PCG_CLKB_O	Precision Clock B Output
11110 (0x1E)	LOW	Logic Level Low (0)
11111 (0x1F)	HIGH	Logic Level High (1)

Serial Data Routing Registers (SRU_DATx, Group B)

The serial data routing control registers (see [Figure A-58](#) through [Figure A-64](#)) route serial data to the serial ports (A and B data channels) and the input data port. Each of the data inputs specified are connected to a data source based on the 6-bit values shown in [Table A-76](#).

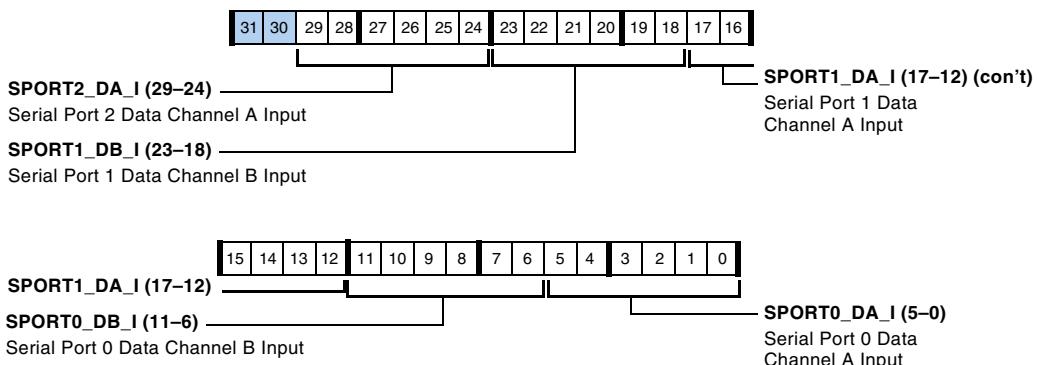


Figure A-58. SRU_DAT0 Register (RW)

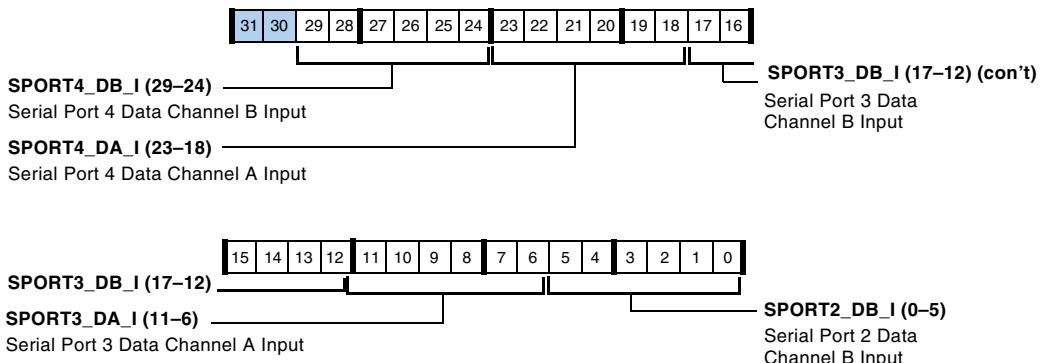


Figure A-59. SRU_DAT1 Register (RW)

DAI Signal Routing Unit Registers

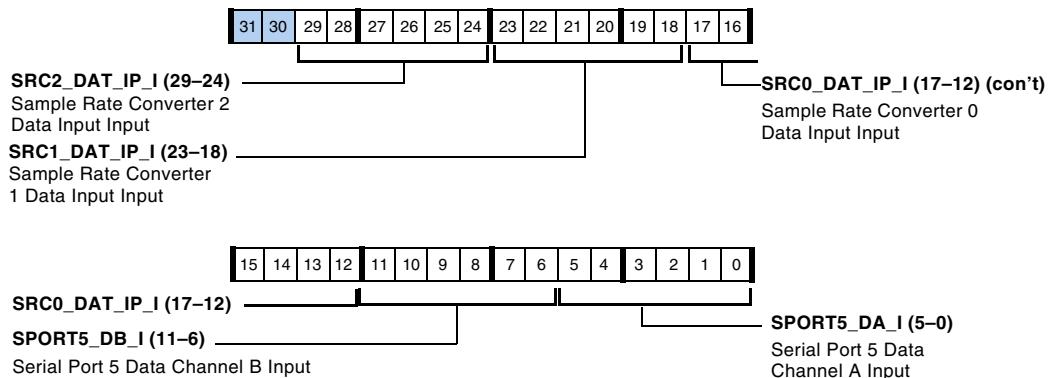


Figure A-60. SRU_DAT2 Register (RW)

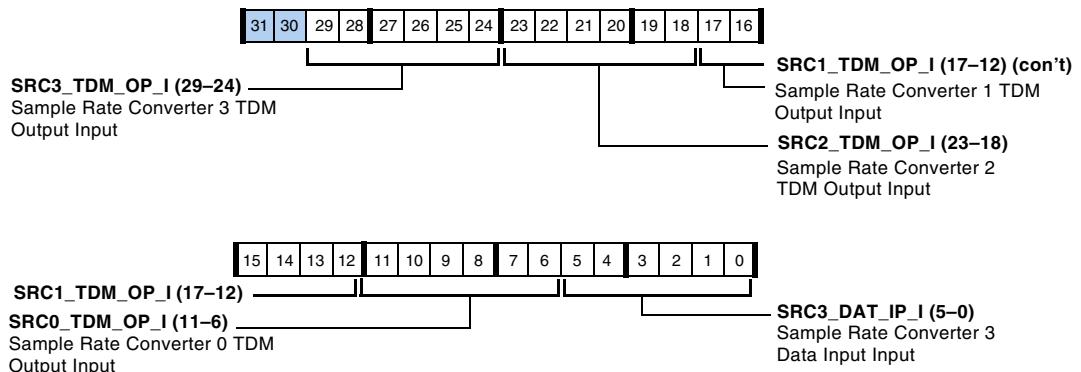


Figure A-61. SRU_DAT3 Register (RW)

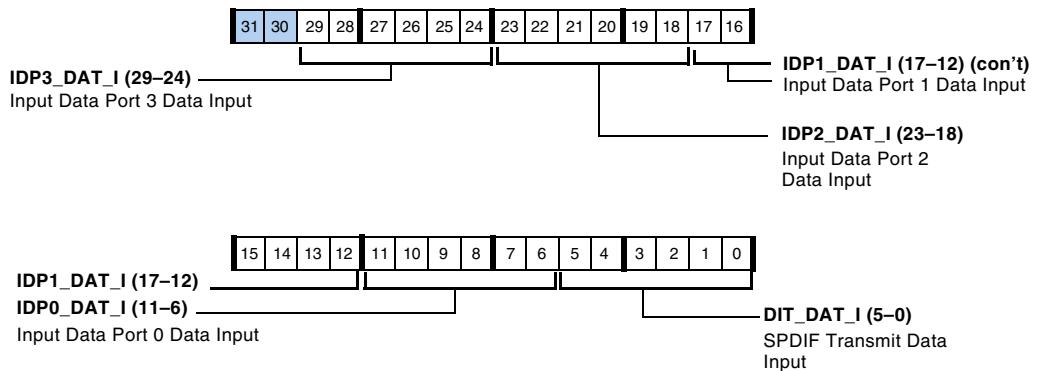


Figure A-62. SRU_DAT4 Register (RW)

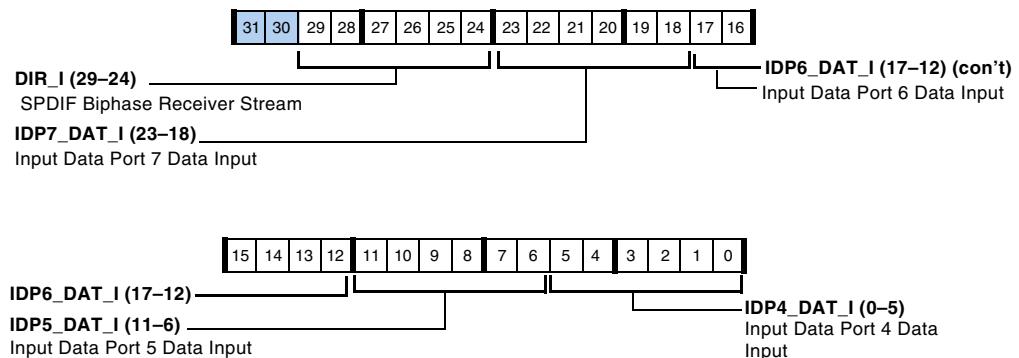


Figure A-63. SRU_DAT5 Register (RW)

DAI Signal Routing Unit Registers

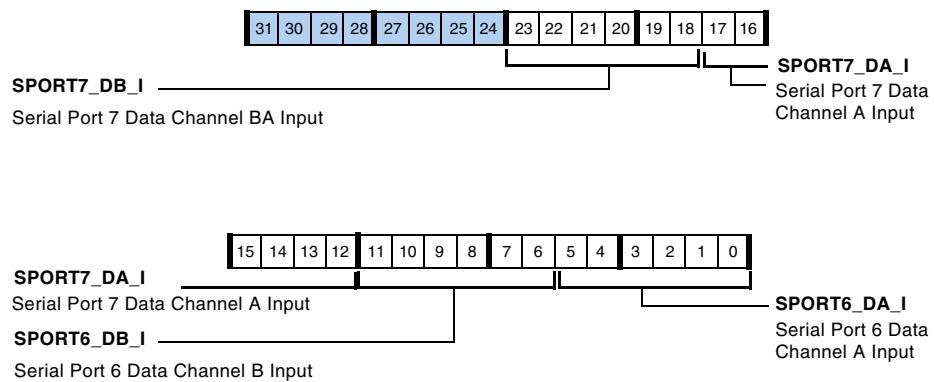


Figure A-64. SRU_DAT6 Register (RW)

Table A-76. Group B Sources – Serial Data

Selection Code	Source Signal	Description (Source Selection)
000000 (0x0)	DAI_PB01_O	Pin Buffer 1
000001 (0x1)	DAI_PB02_O	Pin Buffer 2
000010 (0x2)	DAI_PB03_O	Pin Buffer 3
000011 (0x3)	DAI_PB04_O	Pin Buffer 4
000100 (0x4)	DAI_PB05_O	Pin Buffer 5
000101 (0x5)	DAI_PB06_O	Pin Buffer 6
000110 (0x6)	DAI_PB07_O	Pin Buffer 7
000111 (0x7)	DAI_PB08_O	Pin Buffer 8
001000 (0x8)	DAI_PB09_O	Pin Buffer 9
001001 (0x9)	DAI_PB10_O	Pin Buffer 10
001010 (0xA)	DAI_PB11_O	Pin Buffer 11
001011 (0xB)	DAI_PB12_O	Pin Buffer 12
001100 (0xC)	DAI_PB13_O	Pin Buffer 13
001101 (0xD)	DAI_PB14_O	Pin Buffer 14
001110 (0xE)	DAI_PB15_O	Pin Buffer 15

Table A-76. Group B Sources – Serial Data (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
001111 (0xF)	DAI_PB16_O	Pin Buffer 16
010000 (0x10)	DAI_PB17_O	Pin Buffer 17
010001 (0x11)	DAI_PB18_O	Pin Buffer 18
010010 (0x12)	DAI_PB19_O	Pin Buffer 19
010011 (0x13)	DAI_PB20_O	Pin Buffer 20
010100 (0x14)	SPORT0_DA_O	SPORT 0A Data
010101 (0x15)	SPORT0_DB_O	SPORT 0B Data
010110 (0x16)	SPORT1_DA_O	SPORT 1A Data
010111 (0x17)	SPORT1_DB_O	SPORT 1B Data
011000 (0x18)	SPORT2_DA_O	SPORT 2A Data
011001 (0x19)	SPORT2_DB_O	SPORT 2B Data
011010 (0x1A)	SPORT3_DA_O	SPORT 3A Data
011011 (0x1B)	SPORT3_DB_O	SPORT 3B Data
011100 (0x1C)	SPORT4_DA_O	SPORT 4A Data
011101 (0x1D)	SPORT4_DB_O	SPORT 4B Data
011110 (0x1E)	SPORT5_DA_O	SPORT 5A Data
011111 (0x1F)	SPORT5_DB_O	SPORT 5B Data
100000 (0x20)	SRC0_DAT_OP_O	SRC0 Data Out
100001 (0x21)	SRC1_DAT_OP_O	SRC1 Data Out
100010 (0x22)	SRC2_DAT_OP_O	SRC2 Data Out
100011 (0x23)	SRC3_DAT_OP_O	SRC3 Data Out
100100 (0x24)	SRC0_TDM_IP_O	SRC0 Data Out
100101 (0x25)	SRC1_TDM_IP_O	SRC1 Data Out
100110 (0x26)	SRC2_TDM_IP_O	SRC2 Data Out
100111 (0x27)	SRC3_TDM_IP_O	SRC3 Data Out
101000 (0x28)	DIR_DAT_O	SPDIF RX Serial Data Out

DAI Signal Routing Unit Registers

Table A-76. Group B Sources – Serial Data (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
101100(0x2C)	SPORT6_DA_O	SPORT 6A Data
101101(0x2D)	SPORT6_DB_O	SPORT 6B Data
101110(0x2E)	SPORT7_DA_O	SPORT 7A Data
101111(0x2F)	SPORT7_DB_O	SPORT 7B Data
110000(0x30)	DIT_O	SPDIF TX BiphaseStream
110001(0x31)–111101(0x3D)	Reserved	
111110 (0x3E)	LOW	Logic Level Low (0)
111111 (0x3F)	HIGH	Logic Level High (1)

Frame Sync Routing Control Registers (SRU_F\$X, Group C)

The frame sync routing control registers (see [Figure A-65](#) through [Figure A-69](#)) route a frame sync or a word clock to the serial ports, the SRC, the S/PDIF, and the IDP. Each frame sync input is connected to a frame sync source based on the 5-bit values described in the group C frame sync sources, (listed in [Table A-77](#)).



SPORTs 6 and 7 receive their frame syncs from other routed sources but cannot route their own frame syncs to other SPORTs or other peripherals internally through SRU. If externally needed, they have to be routed through the DAI pins.

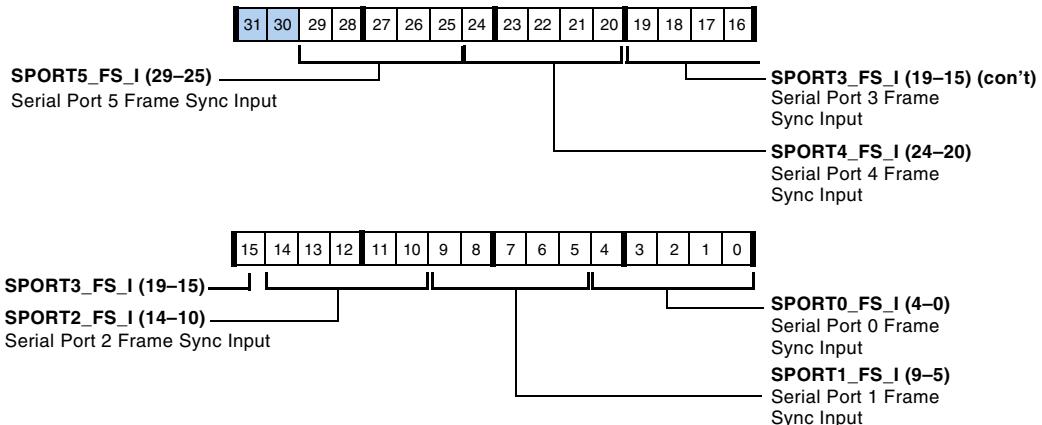


Figure A-65. SRU_FS0 Register (RW)

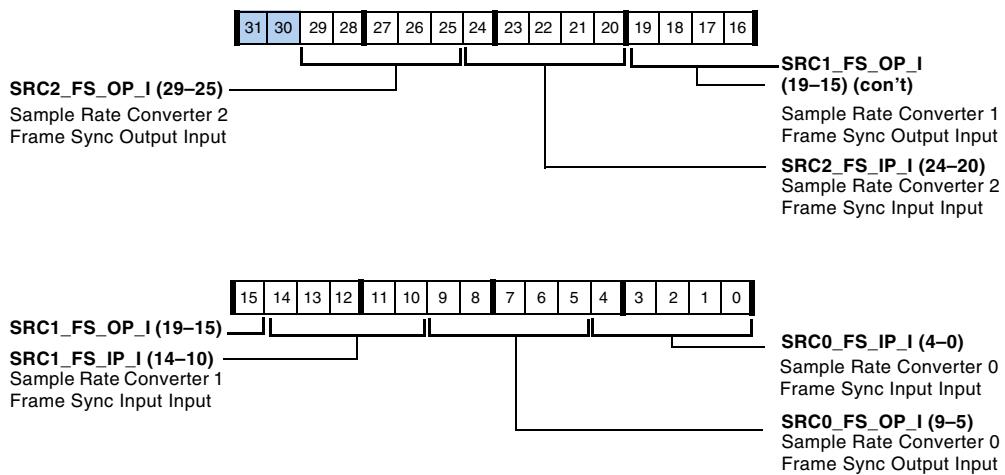


Figure A-66. SRU_FS1 Register (RW)

DAI Signal Routing Unit Registers

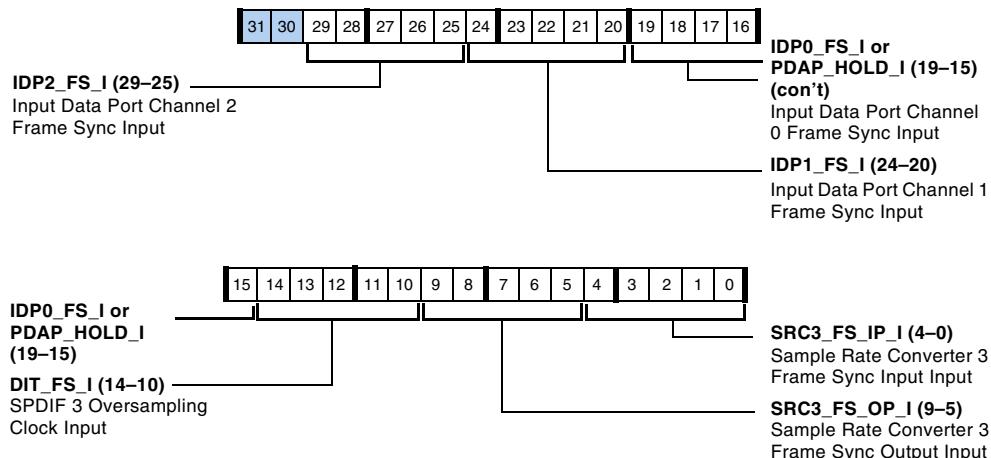


Figure A-67. SRU_FS2 Register (RW)

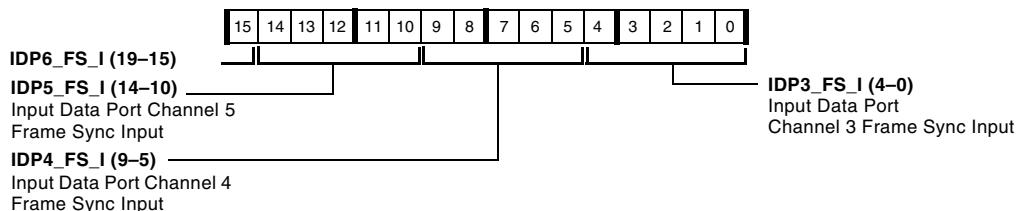
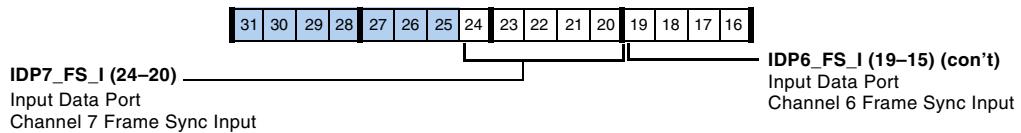


Figure A-68. SRU_FS3 Register (RW)

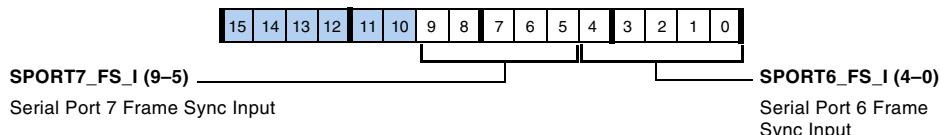


Figure A-69. SRU_FS4 Register (RW)

Table A-77. Group C Sources – Frame Sync

Selection Code	Source Signal	Description (Source Selection)
00000 (0x0)	DAI_PB01_O	Pin Buffer 1
00001 (0x1)	DAI_PB02_O	Pin Buffer 2
00010 (0x2)	DAI_PB03_O	Pin Buffer 3
00011 (0x3)	DAI_PB04_O	Pin Buffer 4
00100 (0x4)	DAI_PB05_O	Pin Buffer 5
00101 (0x5)	DAI_PB06_O	Pin Buffer 6
00110 (0x6)	DAI_PB07_O	Pin Buffer 7
00111 (0x7)	DAI_PB08_O	Pin Buffer 8
01000 (0x8)	DAI_PB09_O	Pin Buffer 9
01001 (0x9)	DAI_PB10_O	Pin Buffer 10
01010 (0xA)	DAI_PB11_O	Pin Buffer 11
01011 (0xB)	DAI_PB12_O	Pin Buffer 12
01100 (0xC)	DAI_PB13_O	Pin Buffer 13
01101 (0xD)	DAI_PB14_O	Pin Buffer 14
01110 (0xE)	DAI_PB15_O	Pin Buffer 15
01111 (0xF)	DAI_PB16_O	Pin Buffer 16
10000 (0x10)	DAI_PB17_O	Pin Buffer 17
10001 (0x11)	DAI_PB18_O	Pin Buffer 18
10010 (0x12)	DAI_PB19_O	Pin Buffer 19
10011 (0x13)	DAI_PB20_O	Pin Buffer 20
10100 (0x14)	SPORT0_FS_O	SPORT 0 Frame Sync
10101 (0x15)	SPORT1_FS_O	SPORT 1 Frame Sync
10110 (0x16)	SPORT2_FS_O	SPORT 2 Frame Sync
10111 (0x17)	SPORT3_FS_O	SPORT 3 Frame Sync
11000 (0x18)	SPORT4_FS_O	SPORT 4 Frame Sync
11001 (0x19)	SPORT5_FS_O	SPORT 5 Frame Sync

DAI Signal Routing Unit Registers

Table A-77. Group C Sources – Frame Sync (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
11010 (0x1A)	DIR_FS_O	SPDIF RX Frame Sync Output
11011 (0x1B)	Reserved	
11100 (0x1C)	PCG_FSA_O	Precision Frame Sync A Output
11101 (0x1D)	PCG_FSB_O	Precision Frame Sync B Output
11110 (0x1E)	LOW	Logic Level Low (0)
11111 (0x1F)	HIGH	Logic Level High (1)

Pin Signal Assignment Registers (SRU_PINx, Group D)

Each physical pin (connected to a bonded pad) may be routed using the pin signal assignment registers (see [Figure A-70](#) through [Figure A-74](#)) in the SRU to any of the inputs or outputs of the DAI peripherals, based on the 7-bit values listed in [Table A-78](#). The SRU also may be used to route signals that control the pins in other ways.

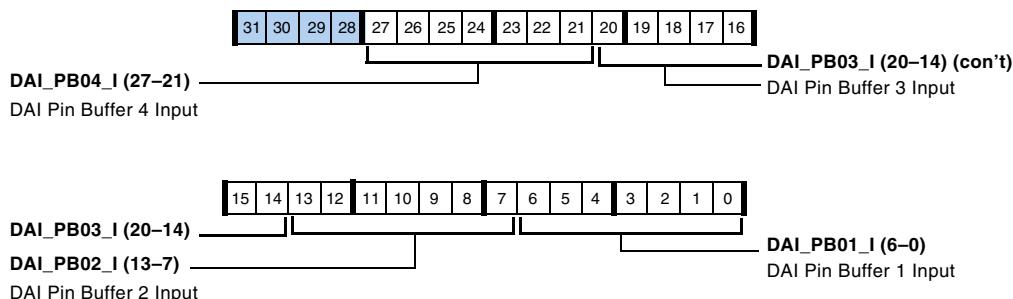


Figure A-70. SRU_PIN0 Register (RW)

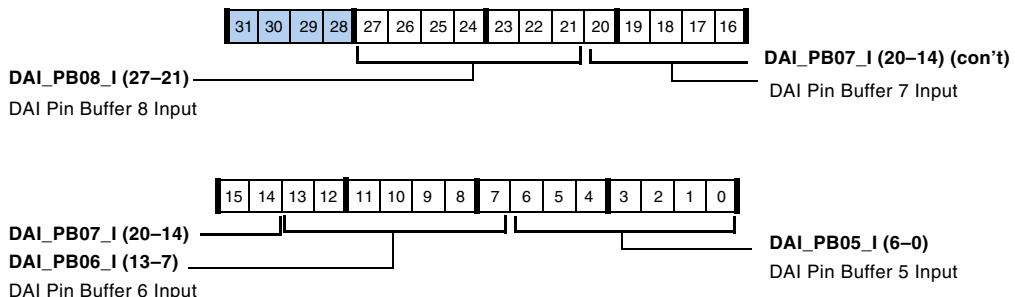


Figure A-71. SRU_PIN1 Register (RW)

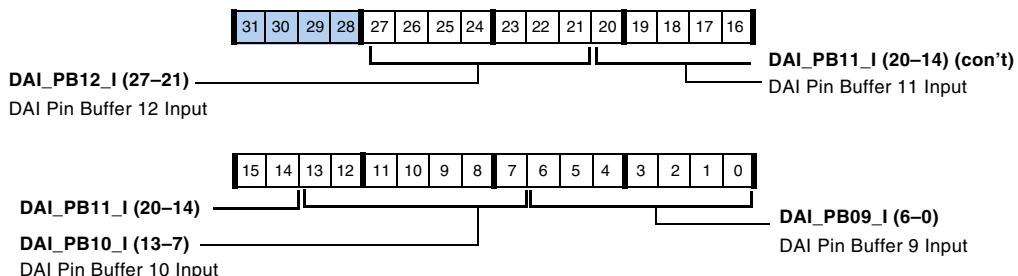


Figure A-72. SRU_PIN2 Register (RW)

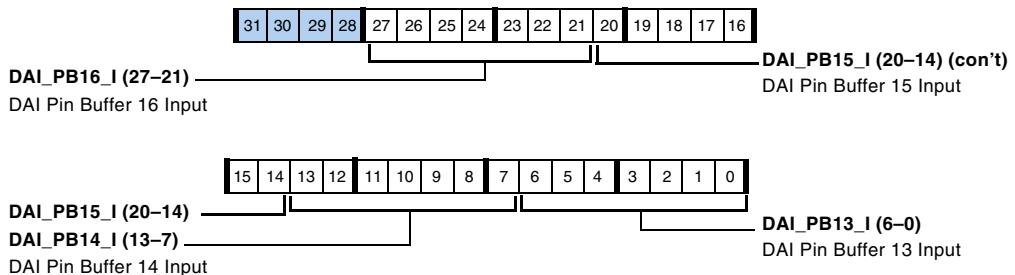


Figure A-73. SRU_PIN3 Register (RW)

DAI Signal Routing Unit Registers

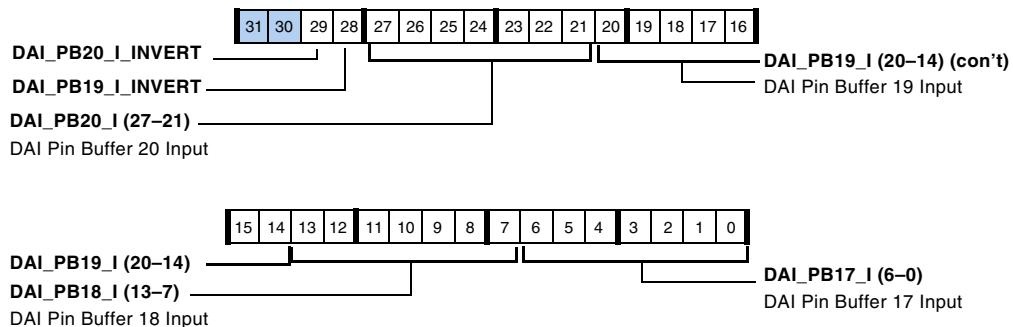


Figure A-74. SRU_PIN4 Register (RW)

Setting SRU_PIN4 bit 28 to high inverts the level of DAI_PB19_I and setting SRU_PIN4 bit 29 to high inverts the level of DAI_PB20_I. Input Inversion only works if the buffer output is not routed to its input.

Table A-78. Group D Sources – Pin Signal Assignments

Selection Code	Source Signal	Description (Source Selection)
0000000 (0x0)	DAI_PB01_O	Pin Buffer 1
0000001 (0x1)	DAI_PB02_O	Pin Buffer 2
0000010 (0x2)	DAI_PB03_O	Pin Buffer 3
0000011 (0x3)	DAI_PB04_O	Pin Buffer 4
0000100 (0x4)	DAI_PB05_O	Pin Buffer 5
0000101 (0x5)	DAI_PB06_O	Pin Buffer 6
0000110 (0x6)	DAI_PB07_O	Pin Buffer 7
0000111 (0x7)	DAI_PB08_O	Pin Buffer 8
0001000 (0x8)	DAI_PB09_O	Pin Buffer 9
0001001 (0x9)	DAI_PB10_O	Pin Buffer 10
0001010 (0xA)	DAI_PB11_O	Pin Buffer 11
0001011 (0xB)	DAI_PB12_O	Pin Buffer 12
0001100 (0xC)	DAI_PB13_O	Pin Buffer 13

Table A-78. Group D Sources – Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
0001101 (0xD)	DAI_PB14_O	Pin Buffer 14
0001110 (0xE)	DAI_PB15_O	Pin Buffer 15
0001111 (0xF)	DAI_PB16_O	Pin Buffer 16
0010000 (0x10)	DAI_PB17_O	Pin Buffer 17
0010001 (0x11)	DAI_PB18_O	Pin Buffer 18
0010010 (0x12)	DAI_PB19_O	Pin Buffer 19
0010011 (0x13)	DAI_PB20_O	Pin Buffer 20
0010100 (0x14)	SPORT0_DA_O	SPORT 0A Data
0010101 (0x15)	SPORT0_DB_O	SPORT 0B Data
0010110 (0x16)	SPORT1_DA_O	SPORT 1A Data
0010111 (0x17)	SPORT1_DB_O	SPORT 1B Data
0011000 (0x18)	SPORT2_DA_O	SPORT 2A Data
0011001 (0x19)	SPORT2_DB_O	SPORT 2B Data
0011010 (0x1A)	SPORT3_DA_O	SPORT 3A Data
0011011 (0x1B)	SPORT3_DB_O	SPORT 3B Data
0011100 (0x1C)	SPORT4_DA_O	SPORT 4A Data
0011101 (0x1D)	SPORT4_DB_O	SPORT 4B Data
0011110 (0x1E)	SPORT5_DA_O	SPORT 5A Data
0011111 (0x1F)	SPORT5_DB_O	SPORT 5B Data
0100000 (0x20)	SPORT0_CLK_O	SPORT 0 Clock
0100001 (0x21)	SPORT1_CLK_O	SPORT 1 Clock
0100010 (0x22)	SPORT2_CLK_O	SPORT 2 Clock
0100011 (0x23)	SPORT3_CLK_O	SPORT 3 Clock
0100100 (0x24)	SPORT4_CLK_O	SPORT 4 Clock
0100101 (0x25)	SPORT5_CLK_O	SPORT 5 Clock
0100110 (0x26)	SPORT0_FS_O	SPORT 0 Frame Sync

DAI Signal Routing Unit Registers

Table A-78. Group D Sources – Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
0100111 (0x27)	SPORT1_FS_O	SPORT 1 Frame Sync
0101000 (0x28)	SPORT2_FS_O	SPORT 2 Frame Sync
0101001 (0x29)	SPORT3_FS_O	SPORT 3 Frame Sync
0101010 (0x2A)	SPORT4_FS_O	SPORT 4 Frame Sync
0101011 (0x2B)	SPORT5_FS_O	SPORT 5 Frame Sync
0101100 (0x2C)	SPORT6_DA_O	SPORT 6A Data
0101101 (0x2D)	SPORT6_DB_O	SPORT 6B Data
0101110 (0x2E)	SPORT7_DA_O	SPORT 7A Data
0101111 (0x2F)	SPORT7_DB_O	SPORT 7B Data
0110000 (0x30)	PDAP_STRB_O	PDAP Data Transfer Request Strobe
0110001 (0x31)	DIT_BLKSTART_O	S/PDIF TX Block Start Output
0110100 (0x34)	SPORT6_CLK_O	SPORT 6 Clock
0110101 (0x35)	SPORT7_CLK_O	SPORT 7 Clock
0110110 (0x36)	SPORT6_FS_O	SPORT 6 Frame Sync
0110111 (0x37)	SPORT7_FS_O	SPORT 7 Frame Sync
0111000 (0x38)	PCG_CLKA_O	Precision Clock A
0111001 (0x39)	PCG_CLKB_O	Precision Clock B
0111010 (0x3A)	PCG_FSA_O	Precision Frame Sync A
0111011 (0x3B)	PCG_FSB_O	Precision Frame Sync B
0111100 (0x3C)	Reserved	
0111101 (0x3D)	SRC0_DAT_OP_O	SRC0 Data Output
0111110 (0x3E)	SRC1_DAT_OP_O	SRC1 Data Output
0111111 (0x3F)	SRC2_DAT_OP_O	SRC2 Data Output
1000000 (0x40)	SRC3_DAT_OP_O	SRC3 Data Output
1000001 (0x41)	DIR_DAT_O	SPDIF_RX Data Output
1000010 (0x42)	DIR_FS_O	SPDIF_RX Frame Sync Output

Table A-78. Group D Sources – Pin Signal Assignments (Cont'd)

Selection Code	Source Signal	Description (Source Selection)
1000011 (0x43)	DIR_CLK_O	SPDIF_RX Clock Output
1000100 (0x44)	DIR_TDMCLK_O	SPDIF_RX TDM Clock Output
1000101 (0x45)	DIT_O	SPDIF TX Biphase Encoded Data Output
1000110 (0x46)	SPORT0_TDVO_O	SPORT0 Transmit Data Valid Output
1000111 (0x47)	SPORT1_TDVO_O	SPORT1 Transmit Data Valid Output
1001000 (0x48)	SPORT2_TDVO_O	SPORT2 Transmit Data Valid Output
1001001 (0x49)	SPORT3_TDVO_O	SPORT3 Transmit Data Valid Output
1001010 (0x4A)	SPORT4_TDVO_O	SPORT4 Transmit Data Valid Output
1001011 (0x4B)	SPORT5_TDVO_O	SPORT5 Transmit Data Valid Output
1001100 (0x4C)	SPORT6_TDVO_O	SPORT6 Transmit Data Valid Output
1001101 (0x4D)	SPORT7_TDVO_O	SPORT7 Transmit Data Valid Output
1001110 (0x4E)	DIR_LRCLK_REF_O	External PLL – Reference Point Connection
1001111 (0x4F)	DIR_LRCLK_FB_O	External PLL – Feedback Point Connection
1010000 (0x50)	PCG_CLKC_O	Precision Clock C
1011001 (0x51)	PCG_CLKD_O	Precision Clock D
1011010 (0x52)	PCG_FSC_O	Precision Frame Sync C
1010011 (0x53)	PCG_FSD_O	Precision Frame Sync D
1010100 – 1111101	Reserved	
1111110 (0x7E)	LOW	Logic Level Low (0)
1111111 (0x7F)	HIGH	Logic Level High (1)

Miscellaneous Signal Routing Registers (SRU_MISCx, Group E)

Miscellaneous register A allows programs to route to the DAI interrupt latch, PBEN input routing, or input signal inversion. In contrast, miscellaneous register B allows programs to only route to the DAI interrupt latch (see [Figure A-75](#) and [Figure A-76](#)).

Notice that when the PCG's one shot option (PCG_PW register) is enabled, the inputs MISCA2_I (PCG unit A) and MISCA3_I (PCG unit B) and MISCA4_I (PCG unit C) and MISCA5_I (PCG unit D) are used as input signals.

The miscellaneous signal routing registers correspond to the group E miscellaneous signals, listed in [Table A-79](#).

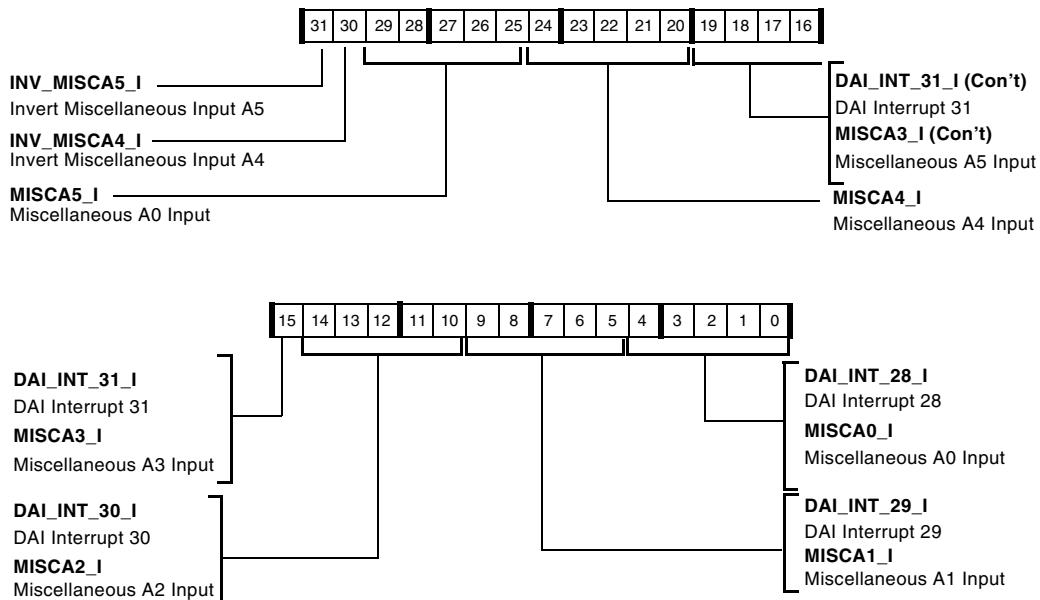


Figure A-75. SRU_EXT_MISC A Register (RW)

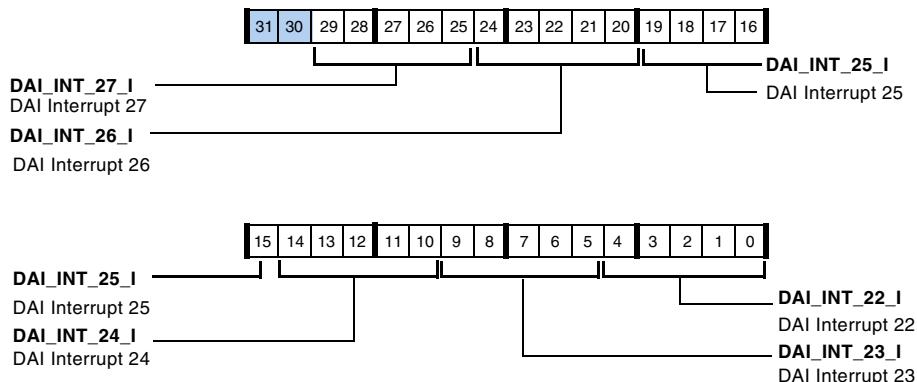


Figure A-76. SRU_EXT_MISCB Register (RW)

Table A-79. Group E Sources – Miscellaneous Signals

Selection Code	Source Signal	Description (Output Source Selection)
00000 (0x0)	DAI_PB01_O	Pin Buffer 1 Output
00001 (0x1)	DAI_PB02_O	Pin Buffer 2 Output
00010 (0x2)	DAI_PB03_O	Pin Buffer 3 Output
00011 (0x3)	DAI_PB04_O	Pin Buffer 4 Output
00100 (0x4)	DAI_PB05_O	Pin Buffer 5 Output
00101 (0x5)	DAI_PB06_O	Pin Buffer 6 Output
00110 (0x6)	DAI_PB07_O	Pin Buffer 7 Output
00111 (0x7)	DAI_PB08_O	Pin Buffer 8 Output
01000 (0x8)	DAI_PB09_O	Pin Buffer 9 Output
01001 (0x9)	DAI_PB10_O	Pin Buffer 10 Output
01010 (0xA)	DAI_PB11_O	Pin Buffer 11 Output
01011 (0xB)	DAI_PB12_O	Pin Buffer 12 Output
01100 (0xC)	DAI_PB13_O	Pin Buffer 13 Output
01101 (0xD)	DAI_PB14_O	Pin Buffer 14 Output
01110 (0xE)	DAI_PB15_O	Pin Buffer 15 Output

DAI Signal Routing Unit Registers

Table A-79. Group E Sources – Miscellaneous Signals (Cont'd)

Selection Code	Source Signal	Description (Output Source Selection)
01111 (0xF)	DAI_PB16_O	Pin Buffer 16 Output
10000 (0x10)	DAI_PB17_O	Pin Buffer 17 Output
10001 (0x11)	DAI_PB18_O	Pin Buffer 18 Output
10010 (0x12)	DAI_PB19_O	Pin Buffer 19 Output
10011 (0x13)	DAI_PB20_O	Pin Buffer 20 Output
10100 (0x14)	SPORT0_FS_O	SPORT0 Frame Sync
10101 (0x15)	SPORT1_FS_O	SPORT1 Frame Sync
10110 (0x16)	SPORT2_FS_O	SPORT2 Frame Sync
10111 (0x17)	SPORT3_FS_O	SPORT3 Frame Sync
11000 (0x18)	SPORT4_FS_O	SPORT4 Frame Sync
11001 (0x19)	SPORT5_FS_O	SPORT5 Frame Sync
11010 (0x1A)	DIT_BLKSTART_O	S/PDIF TX Block Start Output
11011 (0x1B)	PCG_FSA_O	Precision Frame Sync A
11100 (0x1C)	PCG_CLKB_O	Precision Clock B
11101 (0x1D)	PCG_FSB_O	Precision Frame Sync B
11110 (0x1E)	LOW	Logic Level Low (0) as a Source
11111 (0x1F)	HIGH	Logic Level High (1) as a Source

DAI Pin Buffer Enable Registers (SRU_PBENx, Group F)

The pin enable control registers (see [Figure A-77](#) through [Figure A-80](#), [Table A-80](#)) activate the drive buffer for each of the 20 DAI pins. When the pins are not enabled (driven), they can be used as inputs.

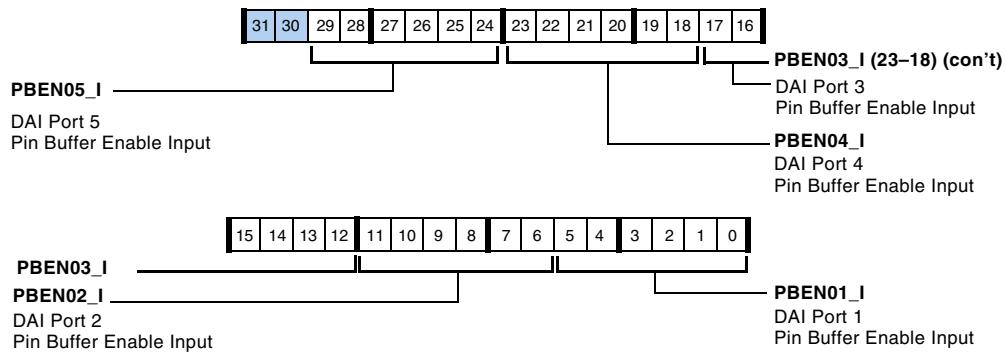


Figure A-77. SRU_PBEN0 (RW)

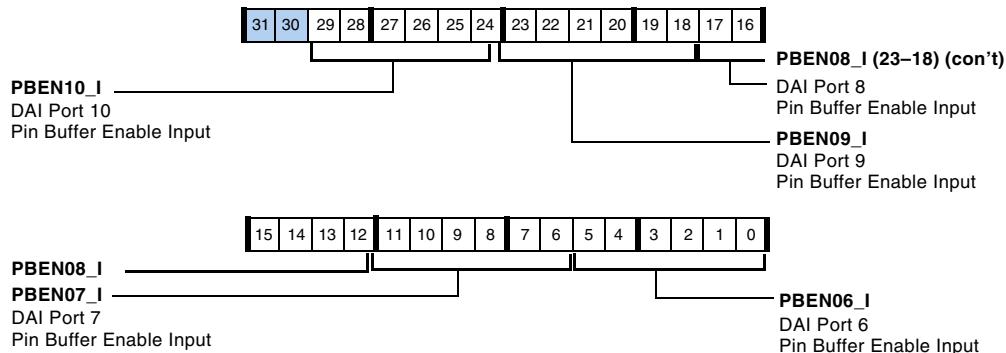


Figure A-78. SRU_PBEN1 (RW)

DAI Signal Routing Unit Registers

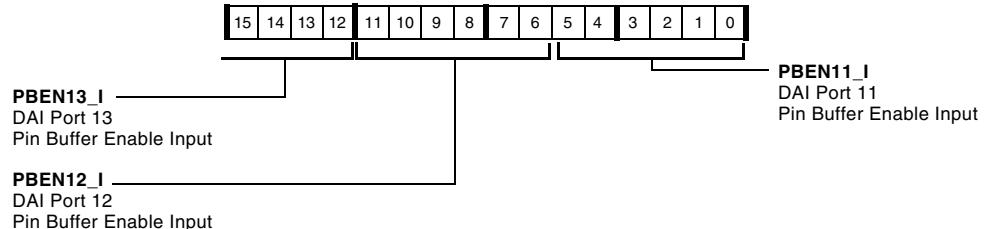
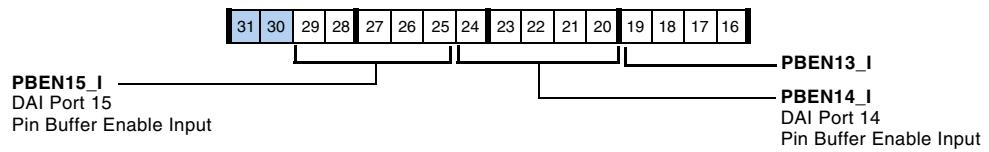


Figure A-79. SRU_PBEN2 (RW)

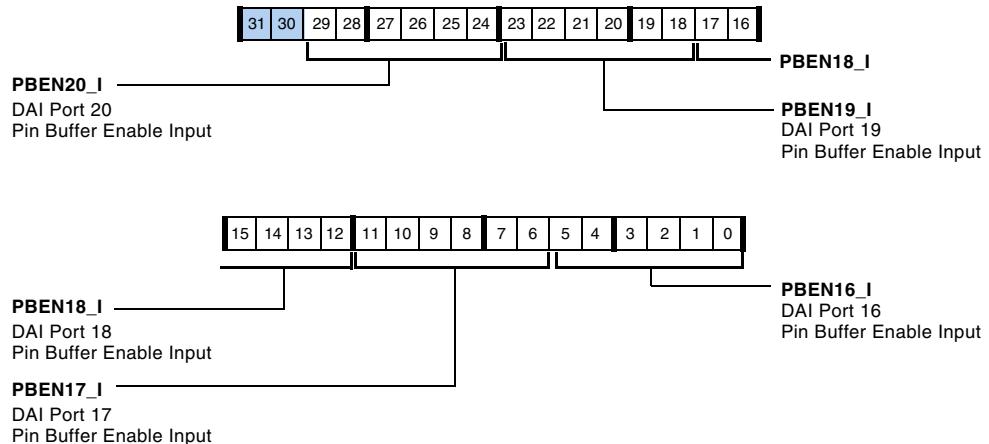


Figure A-80. SRU_PBEN3 (RW)

Table A-80. Group F Sources – Pin Output Enable

Selection Code	Source Signal	Description (Output Source Selection)
000000 (0x0)	LOW	Logic Level Low (0)
000001 (0x1)	HIGH	Logic Level High (1)
000010 (0x2)	MISCA0_O	Miscellaneous Control A0 Output
000011 (0x3)	MISCA1_O	Miscellaneous Control A1 Output
000100 (0x4)	MISCA2_O	Miscellaneous Control A2 Output
000101 (0x5)	MISCA3_O	Miscellaneous Control A3 Output
000110 (0x6)	MISCA4_O	Miscellaneous Control A4 Output
000111 (0x7)	MISCA5_O	Miscellaneous Control A5 Output
001000 (0x8)	SPORT0_CLK_PBEN_O	SPORT 0 Clock Output Enable
001001 (0x9)	SPORT0_FS_PBEN_O	SPORT 0 Frame Sync Output Enable
001010 (0xA)	SPORT0_DA_PBEN_O	SPORT 0 Data Channel A Output Enable
001011 (0xB)	SPORT0_DB_PBEN_O	SPORT 0 Data Channel B Output Enable
001100 (0xC)	SPORT1_CLK_PBEN_O	SPORT 1 Clock Output Enable
001101 (0xD)	SPORT1_FS_PBEN_O	SPORT 1 Frame Sync Output Enable
001110 (0xE)	SPORT1_DA_PBEN_O	SPORT 1 Data Channel A Output Enable
001111 (0xF)	SPORT1_DB_PBEN_O	SPORT 1 Data Channel B Output Enable
010000 (0x10)	SPORT2_CLK_PBEN_O	SPORT 2 Clock Output Enable
010001 (0x11)	SPORT2_FS_PBEN_O	SPORT 2 Frame Sync Output Enable
010010 (0x12)	SPORT2_DA_PBEN_O	SPORT 2 Data Channel A Output Enable
010011 (0x13)	SPORT2_DB_PBEN_O	SPORT 2 Data Channel B Output Enable
010100 (0x14)	SPORT3_CLK_PBEN_O	SPORT 3 Clock Output Enable
010101 (0x15)	SPORT3_FS_PBEN_O	SPORT 3 Frame Sync Output Enable
010110 (0x16)	SPORT3_DA_PBEN_O	SPORT 3 Data Channel A Output Enable
010111 (0x17)	SPORT3_DB_PBEN_O	SPORT 3 Data Channel B Output Enable
011000 (0x18)	SPORT4_CLK_PBEN_O	SPORT 4 Clock Output Enable
011001 (0x19)	SPORT4_FS_PBEN_O	SPORT 4 Frame Sync Output Enable

DAI Signal Routing Unit Registers

Table A-80. Group F Sources – Pin Output Enable (Cont'd)

Selection Code	Source Signal	Description (Output Source Selection)
011010 (0x1A)	SPORT4_DA_PBEN_O	SPORT 4 Data Channel A Output Enable
011011 (0x1B)	SPORT4_DB_PBEN_O	SPORT 4 Data Channel B Output Enable
011100 (0x1C)	SPORT5_CLK_PBEN_O	SPORT 5 Clock Output Enable
011101 (0x1D)	SPORT5_FS_PBEN_O	SPORT 5 Frame Sync Output Enable
011110 (0x1E)	SPORT5_DA_PBEN_O	SPORT 5 Data Channel A Output Enable
011111 (0x1F)	SPORT5_DB_PBEN_O	SPORT 5 Data Channel B Output Enable
100000 (0x20)	SPORT6_CLK_PBEN_O	SPORT 6 Clock Output Enable
100001 (0x21)	SPORT6_FS_PBEN_O	SPORT 6 Frame Sync Output Enable
100010 (0x22)	SPORT6_DA_PBEN_O	SPORT 6 Data Channel A Output Enable
100011 (0x23)	SPORT6_DB_PBEN_O	SPORT 6 Data Channel B Output Enable
100100 (0x24)	SPORT7_CLK_PBEN_O	SPORT 7 Clock Output Enable
100101 (0x25)	SPORT7_FS_PBEN_O	SPORT 7 Frame Sync Output Enable
100110 (0x26)	SPORT7_DA_PBEN_O	SPORT 7 Data Channel A Output Enable
100111 (0x27)	SPORT7_DB_PBEN_O	SPORT 7 Data Channel B Output Enable
101000 (0x28)	SPORT0_TDVT_PBEN_O	SPORT 0 Transmit Data Valid Output
101001 (0x29)	SPORT1_TDVT_PBEN_O	SPORT 1 Transmit Data Valid Output
101010 (0x2A)	SPORT2_TDVT_PBEN_O	SPORT 2 Transmit Data Valid Output
101011 (0x2B)	SPORT3_TDVT_PBEN_O	SPORT 3 Transmit Data Valid Output
101100 (0x2C)	SPORT4_TDVT_PBEN_O	SPORT 4 Transmit Data Valid Output
101101 (0x2D)	SPORT5_TDVT_PBEN_O	SPORT 5 Transmit Data Valid Output
101111 (0x2E)	SPORT6_TDVT_PBEN_O	SPORT 6 Transmit Data Valid Output
101110 (0x2F)	SPORT7_TDVT_PBEN_O	SPORT 7 Transmit Data Valid Output
110000 (0x30)–1111111 (0x3F)		Reserved

DAI Shift Register Routing Registers (Group G, ADSP-2147x)

The pin enable control registers (see [Figure A-81](#), [Figure A-82](#), and [Table A-81](#), [Table A-82](#)) activate the drive buffer for each of the 20 DAI pins. When the pins are not enabled (driven), they can be used as inputs.

Clock Routing Register (SRU_CLK_SHREG)

The shift register's SR_SCLK_I and SR_LAT_I input signals can come from either logic 0, logic 1, the SPORT0–7 clock and frame sync signals, the PCG A and PCG B clock and frame sync signals, SR_SCLK_I, SR_LAT, or DAI pin buffers 1–8. [Figure A-81](#) and [Table A-81](#) show the list of sources and programmable options for SR_SCLK_I and SR_LAT_I input signals.

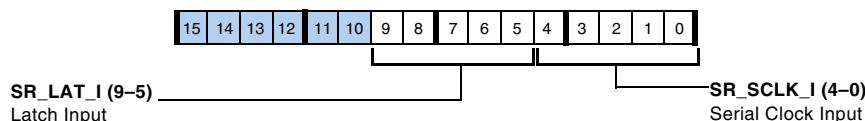


Figure A-81. SR_CLK_SHREG Register (RW)

Table A-81. Group G Sources – Shift Register Clock Routing

Selection Code	Source Signal	Description (Output Source Selection)
00000 (0x0)	LOW	Logic Level Low (0)
00001 (0x1)	HIGH	Logic Level High (1)
00010 (0x2)	SPORT0_CLK_O	Sport 0 Clock Output
00011 (0x3)	SPORT1_CLK_O	Sport 1 Clock Output
00100 (0x4)	SPORT2_CLK_O	Sport 2 Clock Output
00101 (0x5)	SPORT3_CLK_O	Sport 3 Clock Output
00110 (0x6)	SPORT4_CLK_O	Sport 4 Clock Output
00111 (0x7)	SPORT5_CLK_O	Sport 5 Clock Output

DAI Signal Routing Unit Registers

Table A-81. Group G Sources – Shift Register Clock Routing (Cont'd)

Selection Code	Source Signal	Description (Output Source Selection)
01000 (0x8)	SPORT6_CLK_O	Sport 6 Clock Output
01001 (0x9)	SPORT7_CLK_O	Sport 7 Clock Output
01010 (0xA)	SPORT0_FS_O	Sport 0 Frame Sync Output
01011 (0xB)	SPORT1_FS_O	Sport 1 Frame Sync Output
01100 (0xC)	SPORT2_FS_O	Sport 2 Frame Sync Output
01101 (0xD)	SPORT3_FS_O	Sport 3 Frame Sync Output
01110 (0xE)	SPORT4_FS_O	Sport 4 Frame Sync Output
01111 (0xF)	SPORT5_FS_O	Sport 5 Frame Sync Output
10000 (0x10)	SPORT6_FS_O	Sport 6 Frame Sync Output
10001 (0x11)	SPORT7_FS_O	Sport 7 Frame Sync Output
10010 (0x12)	SR_SCLK	External SR_CLK Pin
10011 (0x13)	SR_LAT	External SR_LAT Pin
10100 (0x14)	PCG_CLKA_O	PCG Clock A Output
10101 (0x15)	PCG_CLKB_O	PCG Clock B Output
10110 (0x16)	PCG_FSA_O	PCG Frame Sync A Output
10111 (0x17)	PCG_FSB_O	PCG Frame Sync B Output
11000 (0x18)	DAI_P01_O	DAI External Pin 1
11001 (0x19)	DAI_P02_O	DAI External Pin 2
11010 (0x1A)	DAI_P03_O	DAI External Pin 3
11011 (0x1B)	DAI_P04_O	DAI External Pin 4
11100 (0x1C)	DAI_P05_O	DAI External Pin 5
11101 (0x1D)	DAI_P06_O	DAI External Pin 6
11110 (0x1E)	DAI_P07_O	DAI External Pin 7
11111 (0x1F)	DAI_P08_O	DAI External Pin 8

Data Routing Register (SRU_DAT_SHREG)

The shift register's SR_SDI_I input signal can come from either logic 0 , logic 1, SPORT0–7 data outputs, SR_SDI, or DAI pin buffers 1–8.

[Figure A-82](#) and [Table A-82](#) shows the list of sources and programmable options for the SR_SDI_I input signal.

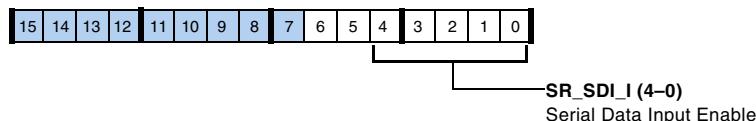


Figure A-82. SRU_DAT_SHREG Register (RW)

Table A-82. Group G Sources – Shift Register Data Routing

Selection Code	Source Signal	Description (Output Source Selection)
00000 (0x0)	LOW	Logic Level Low (0)
00001 (0x1)	HIGH	Logic Level High (1)
00010 (0x2)	SPORT0_DA_O	Sport 0 Data Channel A
00011 (0x3)	SPORT0_DB_O	Sport 0 Data Channel B
00100 (0x4)	SPORT1_DA_O	Sport 1 Data Channel A
00101 (0x5)	SPORT1_DB_O	Sport 1 Data Channel B
00110 (0x6)	SPORT2_DA_O	Sport 2 Data Channel A
00111 (0x7)	SPORT2_DB_O	Sport 2 Data Channel B
01000 (0x8)	SPORT3_DA_O	Sport 3 Data Channel A
01001 (0x9)	SPORT3_DB_O	Sport 3 Data Channel B
01010 (0xA)	SPORT4_DA_O	Sport 4 Data Channel A
01011 (0xB)	SPORT4_DB_O	Sport 4 Data Channel B
01100 (0xC)	SPORT5_DA_O	Sport 5 Data Channel A
01101 (0xD)	SPORT5_DB_O	Sport 5 Data Channel B
01110 (0xE)	SPORT6_DA_O	Sport 6 Data Channel A

DAI Signal Routing Unit Registers

Table A-82. Group G Sources – Shift Register Data Routing (Cont'd)

Selection Code	Source Signal	Description (Output Source Selection)
01111 (0xF)	SPORT6_DB_O	Sport 6 Data Channel B
10000 (0x10)	SPORT7_DA_O	Sport 7 Data Channel A
10001 (0x11)	SPORT7_DB_O	Sport 7 Data Channel B
10010 (0x12)	SR_SDI	External SR_SDI Pin
10011 (0x13)	DAI_P01_O	DAI External Pin 1
10100 (0x14)	DAI_P02_O	DAI External Pin 2
10101 (0x15)	DAI_P03_O	DAI External Pin 3
10110 (0x16)	DAI_P04_O	DAI External Pin 4
10111 (0x17)	DAI_P05_O	DAI External Pin 5
11000 (0x18)	DAI_P06_O	DAI External Pin 6
11001 (0x19)	DAI_P07_O	DAI External Pin 7
11010 (0x1A)	DAI_P08_O	DAI External Pin 8
11011 (0x1B) – 11111 (0x1F)	Reserved	

DAI Pin Buffer Registers

The DAI_STAT register, shown in [Figure A-97](#) and described in [Table A-94](#), and the DAI_PIN_STAT register, shown in [Figure A-83 on page A-149](#), provide status information for the IDP/PDAP DMA channels.

Pin Buffer Registers (DAI_PIN_STAT)

The DAI_PIN_STAT register, shown in [Figure A-83](#), provides DAI pin buffer status information. This register is updated at up to the PCLK/2 rate.

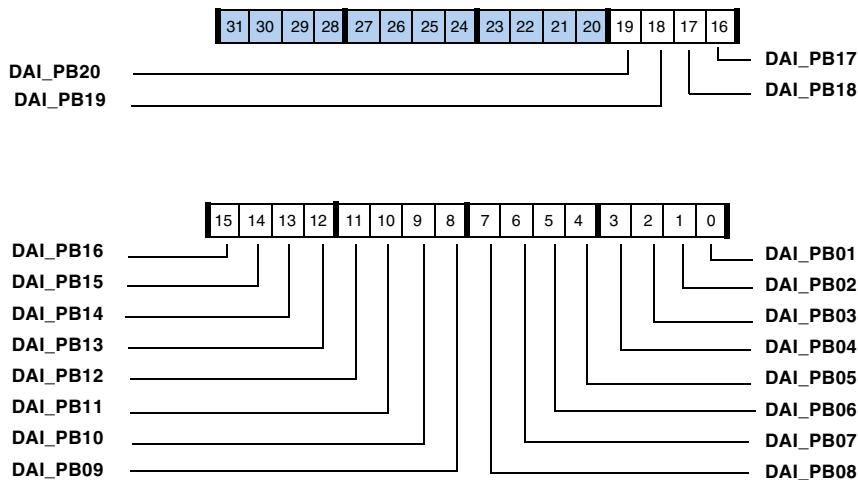


Figure A-83. DAI_PIN_STAT Register (RO)

Interrupt Controller Registers

All of the DAI interrupt registers are used primarily to provide the status of the resident interrupt controller. These registers are listed in [Table A-83](#) and shown in [Figure A-84](#). Note that for each of these registers the bit names and numbers are the same.

Table A-83. DAI Interrupt Registers

Register	Description
DAI_IRPTL_H (ROC)	High priority interrupt latch register
DAI_IRPTL_HS (RO)	Shadow high priority interrupt latch register
DAI_IRPTL_L (ROC)	Low priority interrupt latch register
DAI_IRPTL_LS (RO)	Shadow low priority interrupt latch register
DAI_IMASK_PRI (RW)	Core interrupt priority assignment register
DAI_IMASK_RE (RW)	Rising edge interrupt mask register
DAI_IMASK_FE (RW)	Falling edge interrupt mask register

Peripherals Routed Through the DAI

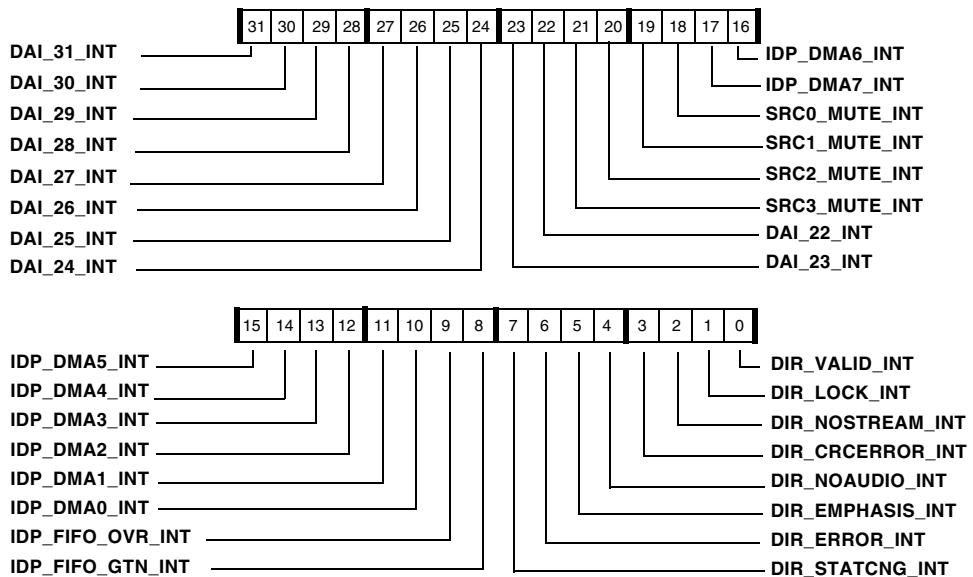


Figure A-84. DAI_IRPTL_x Register

Peripherals Routed Through the DAI

The following sections provide information on the peripherals that are explicitly routed through the digital applications interface. [For more information, see “DAI Signal Routing Unit Registers” on page A-118.](#)

Serial Port Registers

The following section describes serial port (SPORT) registers.

SPIRIT Divisor Registers (DIVx)

These registers, shown in [Figure A-85](#), allow programs to set the frame sync divisor and clock divisor.

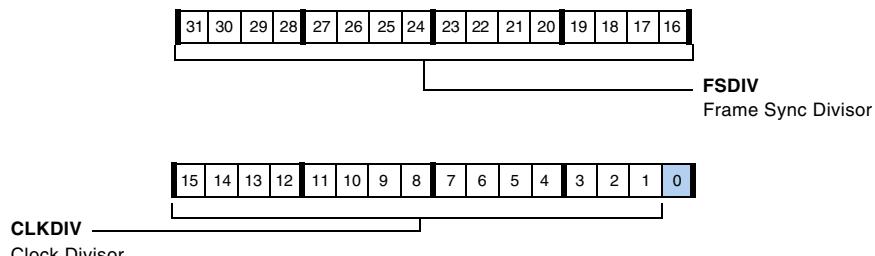


Figure A-85. DIVx Register (RW)

Serial Control Registers (SPCTLx)

The SPCTLx registers are transmit and receive control registers for the corresponding serial ports (SPORT 0 through 7). These registers change depending on operating mode. Figures and bit descriptions are provided as follows.

- Serial mode – [Figure A-86](#) and [Table A-84 on page A-153](#).
- I²S and Left-Justified modes – [Figure A-87 on page A-158](#) and [Table A-85 on page A-159](#).
- Packed and Multichannel mode – [Figure A-88 on page A-162](#) and [Table A-86 on page A-163](#).

Peripherals Routed Through the DAI

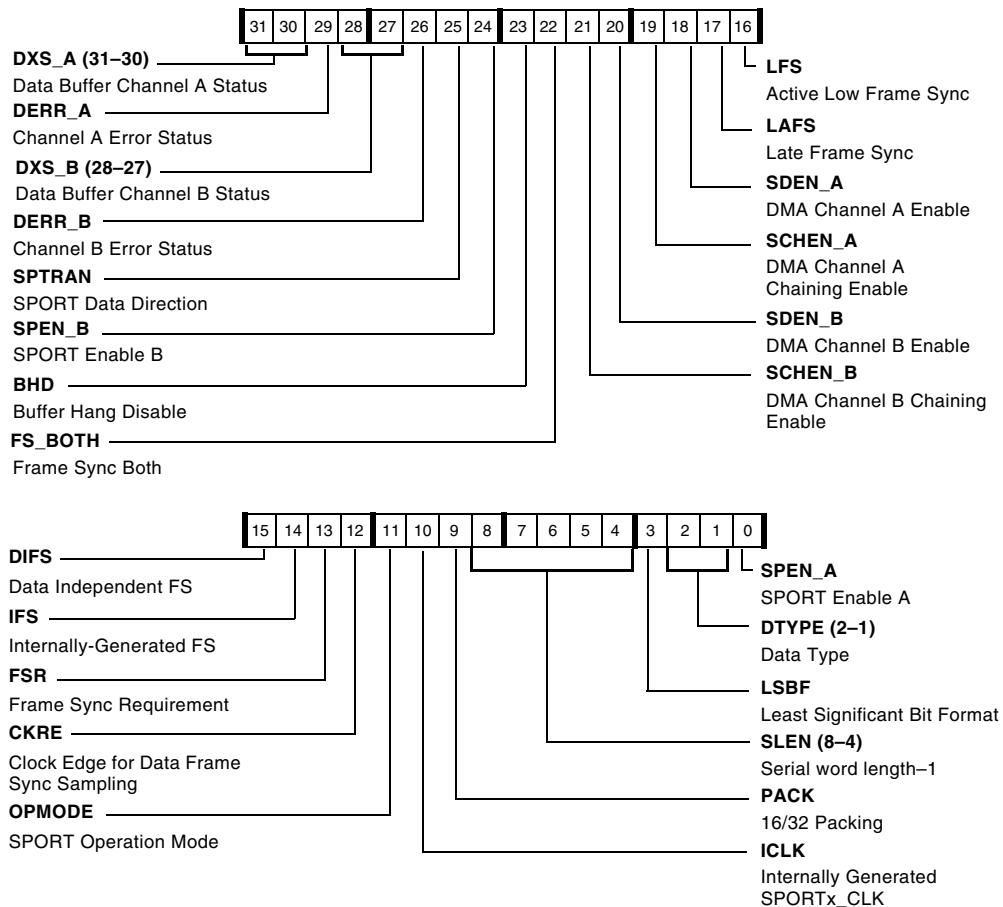


Figure A-86. SPCTLx Register for Standard Serial Mode

Table A-84. SPCTLx Register Bit Descriptions (Standard Serial Mode) (RW)

Bit	Name	Description
0	SPEN_A	Enable Channel A Serial Port. 0 = Serial port A channel disabled 1 = Serial port A channel enabled
2–1	DTYPE	Data Type Select. Selects the data type formatting for standard serial mode transmissions. For standard serial mode A channels, selection of companding mode and MSB format are exclusive: Serial Data Channel A Type Formatting 00 Right-justify, zero-fill unused MSBs 01 Right-justify, sign-extend unused MSBs 10 Compand using μ -law 11 Compand using A-law Serial Data Channel B Type Formatting For Standard serial mode B channels, companding mode is unavailable. 0 Right-justify, zero-fill unused MSBs 1 Right-justify, sign-extend unused MSBs The transmit buffer does not zero-fill or sign-extend transmit data words; this only takes place for the receive buffer.
3	LSBF	Serial Word Endian Select. 0 = Big endian (MSB first) 1 = Little endian (LSB first)
8–4	SLEN	Serial Word Length Select. Selects the word length in bits. The value of SLEN = serial word length. For DSP standard mode, word sizes can range from 3 bit (SLEN = 3) to 32 bits (SLEN = 32). Do not set the SLEN less than 3. Words smaller than 32 bits are right-justified in the receive and transmit buffers, residing in the least significant (LSB) bit positions.

Peripherals Routed Through the DAI

Table A-84. SPCTLx Register Bit Descriptions (Standard Serial Mode) (RW) (Cont'd)

Bit	Name	Description
9	PACK	<p>16-Bit to 32-Bit Word Packing Enable. When PACK = 1, two successive received words are packed into a single 32-bit word, and each 32-bit word is unpacked and transmitted as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15–0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31–16. This applies to both receive (packing) and transmit (unpacking) operations. Companding can be used with word packing or unpacking. When SPORT data packing is enabled, the transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word. Note that when 16-bit received data is packed into 32-bit words and stored in NW-word space in internal memory, the 16-bit words can be read or written with SW-space addresses.</p> <p>0 = Disable 16- to 32-bit word packing 1 = Enable 16- to 32-bit word packing</p>
10	ICLK	<p>Internal Clock Select. When ICLK is set (=1), the clock signal is generated internally by the processor and the SPORTx_CLK_O signals are outputs. The clock frequency is determined by the value of the serial clock divisor (CLKDIV) in the DIVx registers. When ICLK is cleared (=0), the clock signal is accepted as an input on the SPORTx_CLK_I signals, and the serial clock divisors in the DIVx registers are ignored. Note the externally-generated serial clock does not need to be synchronous with the processor's system clock.</p> <p>0 = Select external transmit clock 1 = Select internal transmit clock</p>
11	OPMODE	<p>Sport Operation Mode. 0 = DSP serial/multichannel mode if cleared</p>

Table A-84. SPCTLx Register Bit Descriptions (Standard Serial Mode) (RW) (Cont'd)

Bit	Name	Description
12	CKRE	<p>Clock Rising Edge Select. Determines clock signal to sample data and the frame sync selection. For sampling receive data and frame syncs, setting CKRE to 1 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.</p> <p>For example, the transmit and receive functions of any two SPORTs connected together should always select the same value for CKRE so internally-generated signals are driven on one edge and received signals are sampled on the opposite edge.</p> <p>0 = Falling edge 1 = Rising edge</p>
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).
14	IFS	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).
15	DIFS	<p>Data Independent Frame Sync Select.</p> <p>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).</p>
16	LFS	Active Low Frame Sync Select. 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).
17	LAFS	Late Transmit Frame Sync Select. This bit selects when to generate the frame sync signal. This bit selects a late frame sync if set (= 1) during the first bit of each data word. This bit selects an early frame sync if cleared (= 0) during the serial clock cycle immediately preceding the first data bit
		<p>0 = Early frame sync (FS before first bit) 1 = Late frame sync (FS during first bit)</p>

Peripherals Routed Through the DAI

Table A-84. SPCTLx Register Bit Descriptions (Standard Serial Mode) (RW) (Cont'd)

Bit	Name	Description
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
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. This bit applies when the SPORTS channels A and B are configured to transmit/receive data. If set (= 1), this bit issues frame sync only when data is present in both transmit buffers, TXSPxA and TXSPxB. If cleared (= 0), a frame sync is issued if data is present in either transmit buffers. When a SPORT is configured as a receiver, if FS_BOTH is set (= 1), frame sync is issued only when both the RX FIFOs (RXSPxA and RXSPxB) are not full. 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). 0 = Issue FS if data is present in either transmit buffer. 1 = Issue FS if data is present in both transmit buffers.
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
24	SPEN_B	Enable Channel B Serial Port. 0 = Serial port A channel disabled 1 = Serial port A channel enabled

Table A-84. SPCTLx Register Bit Descriptions (Standard Serial Mode) (RW) (Cont'd)

Bit	Name	Description
25	SPTRAN	<p>Data Direction Control. This bit controls the data direction of the serial port channel A and B signals.</p> <p>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.</p> <p>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.</p>
26 (RO)	DERR_B	<p>Channel B Error Status. When the SPORT is configured as a transmitter, this bit provides transmit underflow status. If FSR = 1, DERR_x bit indicates whether the SPORTx_FS signal (from an internal or external source) occurred while the DXS_x buffer was empty. The SPORTs transmit data whenever they detect a SPORTx_FS signal.</p> <p>0 = No SPORTx_FS signal occurred while TXSPxA/B buffer is empty. 1 = SPORTx_FS signal occurred while TXSPxA/B buffer is empty.</p> <p>If FSR = 0, DERR_x is set whenever the SPORT is required to transmit and the transmit buffer is empty.</p> <p>When the SPORT is configured as a receiver, this bit provides receive overflow status. If FSR = 1, DERR_x bit indicates whether the SPORTx_FS signal (from an internal or external source) occurred while the DXS_x buffer was full. The SPORTs receive data whenever they detect a SPORTx_FS signal. As a receiver, it indicates when the channel has received new data while the receive buffer is full. New data overwrites existing data.</p> <p>0 = No SPORTx_FS signal occurred while RXSPxA/B buffer is full. 1 = SPORTx_FS signal occurred while RXSPxA/B buffer is full.</p> <p>If FSR = 0, DERR_x is set whenever the SPORT is required to receive and the receive buffer is full.</p>
28–27 (RO)	DXS_B	<p>Channel B Data Buffer Status. Indicates the status of the serial port's channel B data buffer (RXSPxB or TXSPxB) as follows:</p> <p>00 = Empty 10 = Partially full 11 = Full</p>

Peripherals Routed Through the DAI

Table A-84. SPCTLx Register Bit Descriptions (Standard Serial Mode) (RW) (Cont'd)

Bit	Name	Description
29 (RO)	DERR_A	Channel A Error Status (sticky). Refer to DERR_B
31–30 (RO)	DXS_A	Channel A Data Buffer Status. Refer to DXS_B

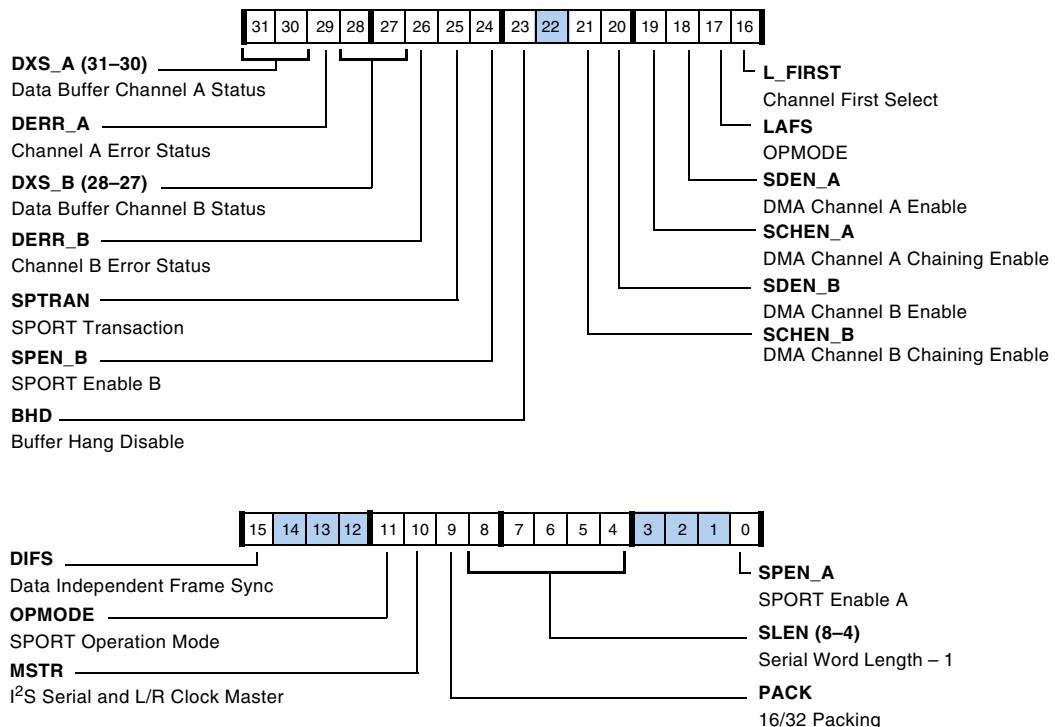


Figure A-87. SPCTLx Register for I²S and Left-Justified Modes

Table A-85. SPCTLx Register Bit Descriptions (I²S, Left-Justified) (RW)

Bit	Name	Description
0	SPEN_A	Enable Channel A Serial Port. 0 = Serial port A channel disabled 1 = Serial port A channel enabled
3–1	Reserved	
8–4	SLEN	Serial Word Length Select. Selects the word length in bits. Word sizes can be from 8 bits to 32 bits.
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	MSTR	Master Clock Select. 0 = Select external clock and WS 1 = Select internal clock and WS
11	OPMODE	Sport Operation Mode. 1 = Selects the I ² S or left-justified mode. Bit 17 is used to select either of both modes.
14–12	Reserved	
15	DIFS	Data Independent Frame Sync Select. 0 = Serial port uses a data-dependent frame sync (sync when TX FIFO is not empty or when RX FIFO is not full). 1 = Serial port uses a data-independent frame sync (sync at selected interval)
16	L_FIRST	Frame Sync Channel First Select. Selects left or right channel word first after valid edge of frame sync. For left-justified mode channel order: 0 = first data after the rising edge 1 = first data after the falling edge For I ² S mode channel order: 0 = first data after the falling edge 1 = first data after the rising edge
17	OPMODE (LAFS)	Operation Mode (I²S or Left-Justified Mode Select). 0 = I ² S mode 1 = Left-justified mode

Peripherals Routed Through the DAI

Table A-85. SPCTLx Register Bit Descriptions (I²S, Left-Justified) (RW) (Cont'd)

Bit	Name	Description
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
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	Reserved	
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
24	SPEN_B	Enable Channel B Serial Port. 0 = Serial port A channel disabled 1 = Serial port A channel enabled
25	SPTRAN	Data Direction Control. This bit controls the data direction of the serial port 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.

Table A-85. SPCTLx Register Bit Descriptions (I²S, Left-Justified) (RW) (Cont'd)

Bit	Name	Description
26 (RO)	DERR_B	<p>Channel B Error Status. SPORT configured as a transmitter, this bit provides transmit underflow status. As a transmitter, DERR_x bit indicates whether the SPORTx_FS signal (from an internal or external source) occurred while the DXS_x buffer was empty. The SPORTs transmit data whenever they detect a SPORTx_FS signal.</p> <p>0 = No SPORTx_FS signal occurred while TXSPxA/B buffer is empty. 1 = SPORTx_FS signal occurred while TXSPxA/B buffer is empty.</p> <p>SPORT configured as a receiver, these bits provide receive overflow status. As a receiver, DERR_x bit indicates whether the SPORTx_FS signal (from an internal or external source) occurred while the DXS_x buffer was full. The SPORTs receive data whenever they detect a SPORTx_FS signal. As a receiver, it indicates when the channel has received new data while the receive buffer is full. New data overwrites existing data.</p> <p>0 = No SPORTx_FS signal occurred while RXSPxA/B buffer is full. 1 = SPORTx_FS signal occurred while RXSPxA/B buffer is full.</p>
28–27 (RO)	DXS_B	<p>Channel B Data Buffer Status. Indicates the status of the SPORT's channel B data buffer (RXSPxB or TXSPxB) as follows:</p> <p>00 = Empty 10 = Partially full 11 = Full</p>
29 (RO)	DERR_A	Channel A Error Status. Refer to DERR_B.
31–30 (RO)	DXS_A	Channel A Data Buffer Status. Refer to DXS_B.

Peripherals Routed Through the DAI

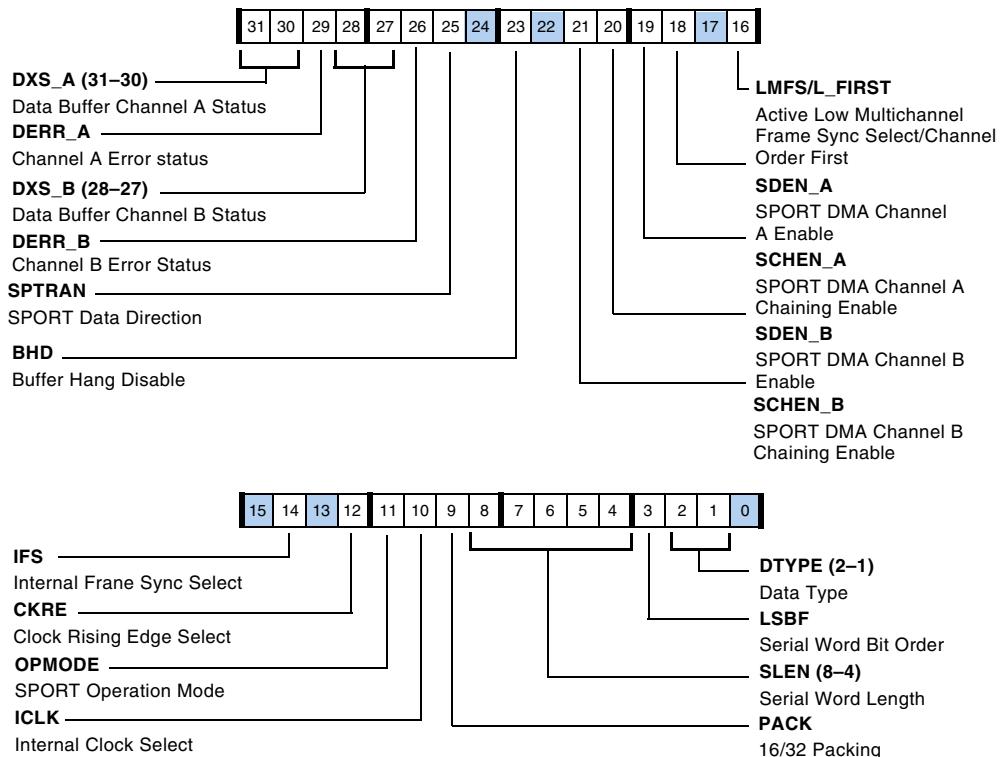


Figure A-88. SPCTLx Register – Packed and Multichannel Mode

Table A-86. SPCTLx Register Bit Descriptions (Packed and Multichannel) (RW)

Bit	Name	Description
0	Reserved	
2–1	DTYPE	<p>Data Type Select. Selects the data type formatting for multichannel/packed mode transmissions. For multichannel/packed mode A channels, selection of companding mode and MSB format are inclusive:</p> <p>Serial Data Channel A Type Formatting</p> <ul style="list-style-type: none"> x0 = Right-justify, zero-fill unused MSBs x1 = Right-justify, sign-extend unused MSBs 1x = Compand using μ-law 1x = Compand using A-law <p>For multichannel/packed mode B channels, companding mode not available.</p> <p>Serial Data Channel B Type Formatting</p> <ul style="list-style-type: none"> 0 = Right-justify, zero-fill unused MSBs 1 = Right-justify, sign-extend unused MSBs <p>The transmit buffer does not zero-fill or sign-extend transmit data words; this only takes place for the receive buffer.</p>
3	LSBF	<p>Serial Word Endian Select.</p> <ul style="list-style-type: none"> 0 = Big endian (MSB first) 1 = Little endian (LSB first)
8–4	SLEN	Serial Word Length Select. Selects the word length in bits. Word sizes can be from 3 bits to 32 bits.
9	PACK	<p>16-bit to 32-bit Word Packing Enable.</p> <ul style="list-style-type: none"> 0 = Disable 16- to 32-bit word packing 1 = Enable 16- to 32-bit word packing
10	ICLK	<p>Internal Clock Select. Select the SPORT clock.</p> <ul style="list-style-type: none"> 0 = Select external clock 1 = Select internal clock
11	OPMODE	<p>Sport Operation Mode.</p> <ul style="list-style-type: none"> 0 = Multichannel mode 1 = Packed mode <p>Note for multichannel operation, the SPMCTLx registers must be programmed.</p>

Peripherals Routed Through the DAI

Table A-86. SPCTLx Register Bit Descriptions (Packed and Multichannel) (RW) (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
13	Reserved	
14	IMFS	Internal Multichannel 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).
15	Reserved	
16	LMFS/L_FIRST	Active Multichannel Frame Sync/Channel Order First. For multi-channel mode this bit (LMFS) selects the logic level of the (transmit or receive) frame sync signals. 0 = Active HIGH level frame sync 1 = Active LOW level frame sync If FSED bit in SPCTLNx register is high (=1) the SPORTs detects an active edge of an external frame sync and starts transmitting/receiving only after that (even if you enable SPORTs at any instant of active frame sync). This is done only when SPORTs are programmed for external FS mode (IMFS = 0). If FSED bit is cleared (reset value), SPORTs behaves similar to previous SHARC processors. For packed mode, this bit (L_FIRST) selects left or right channel word first after valid edge. 0 = Tx/Rx on right channel first 1 = Tx/Rx on left channel first
17	Reserved	
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-86. SPCTLx Register Bit Descriptions (Packed and Multichannel) (RW) (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	Reserved	
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
24	Reserved	
25	SPTRAN	Data Direction Control. This bit controls the data direction of the serial port 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.

Peripherals Routed Through the DAI

Table A-86. SPCTLx Register Bit Descriptions (Packed and Multichannel) (RW) (Cont'd)

Bit	Name	Description
26 (RO, sticky)	DERR_B	<p>Channel B Error Status. When the SPORT is configured as a transmitter, this bit provides transmit underflow status. As a transmitter DERR_x bit indicates whether the SPORTx_FS signal (from an internal or external source) occurred while the DXS_x buffer was empty. The SPORTs transmit data whenever they detect a SPORTx_FS signal.</p> <p>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, DERR_x bit indicates whether the SPORTx_FS signal (from an internal or external source) occurred while the DXS_x buffer was full. The SPORTs receive data whenever they detect a SPORTx_FS signal. As a receiver, it indicates when the channel has received new data while the receive buffer is full. New data overwrites existing data.</p> <p>0 = No SPORTx_FS signal occurred while RXSPxA/B buffer is full. 1 = SPORTx_FS signal occurred while RXSPxA/B buffer is full.</p>
28–27 (RO)	DXS_B	<p>Channel B Data Buffer Status. Indicates the status of the serial port's receive channel B data buffer as follows:</p> <p>00 = Empty 10 = Partially full 11 = Full</p>
29 (RO, sticky)	DERR_A	<p>Channel A Error Status. Indicates if the serial receive operation has overflowed in the channel A data buffer.</p>
31–30 (RO)	DXS_A	<p>Channel A Data Buffer Status. Indicates the status of the serial port's receive channel A data buffer as follows:</p> <p>00 = Empty 10 = Partially full 11 = Full</p>

SPORT Control 2 Registers (SPCTLNx)

These registers (where x signifies SPORT 0 through 7) allow programs to set frame sync edge detection for I²S compatibility. These registers also allow interrupts to be generated when transmit DMA count is expired or when the last bit of last word is shifted out.



Note that these registers do not exist on previous SHARC processors (ADSP-212xx, ADSP-213xx).

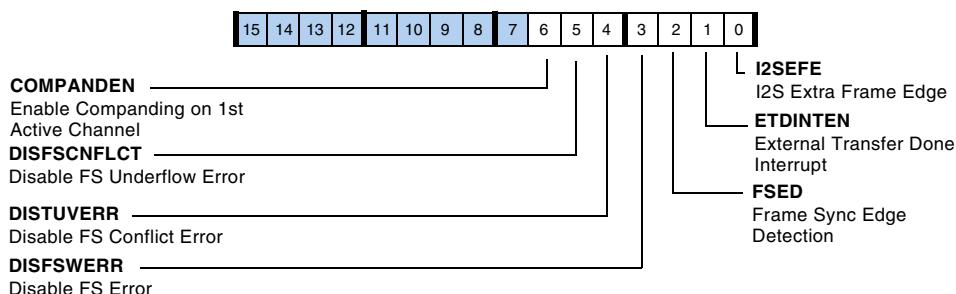


Figure A-89. SPCTLNx Register

Table A-87. SPCTLNx Register Bit Descriptions (RW)

Bit	Name	Description
0	I2SEFE	I2S Extra Frame Edge. If set, SPORT generates the last LRCLK if configured as I2S master (valid only for DMA). If cleared, (reset value), behaves similar to previous SHARCs.
1	ETDINTEN	External Transfer Done Interrupt. If set, interrupt occurs only after the last bit of last word in the DMA is shifter out. If cleared, interrupt occurs when the DMA counter expires. For chain pointer DMA, if set, interrupt occurs: 1) only after that last word of last DMA block in the chain is shifted out, if PCI=0. 2) When DMA counter expires for the initial DMA blocks (CP nonzero) in the chain and the last bit of the last word is shifted out for the last DMA block in the chain, if PCI=1. For RX DMA, interrupt behaves in the same way independent of the value of bit setting.

Peripherals Routed Through the DAI

Table A-87. SPCTLNx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
2	FSED	Frame Sync Edge Detection. If set in multichannel mode, the SPORTs detect an active edge of an external frame sync and start transmitting/receiving only after that (even if the SPORTs are enabled at any instant of active frame sync). This is done only when SPORTs are programmed for external FS mode (IMFS = 0). If cleared (reset value), behaves similar to previous SHARCs.
3	DISFSWERR (Applies to ADSP-2147x, ADSP-2148x)	Disable Frame Sync Error. If an external frame sync is of more than one SCLK width in serial mode, a frame sync error is generated. In late frame sync mode if a frame sync is not active during the whole transmission/reception a frame sync error is generated. If this bit is set, the frame sync error is generated only on an active edge of premature frame sync during valid data transmission/reception. An error is not generated even if the frame sync is of more than one SCLK width or if it is not active throughout the transmit/receive in late frame sync mode.
4	DISTUVERR (Applies to ADSP-2147x, ADSP-2148x)	Disable Underflow Error. If single channel is enabled in multichannel mode, and if a premature frame sync occurs (for example: word length = 16 bits, frame sync duration 14 SCLK) during the transmission of last word in a DMA, then the TX underflow error bit is set (DERR_x bit), even though this premature frame sync does not cause any underflow and the SPORT does not try to drive data for this premature frame sync. If this bit is set, no spurious TX underflow error is generated for this premature frame sync.
5	DISFSCNFLCT (Applies to ADSP-2147x, ADSP-2148x)	Disable Frame Sync Conflict Error. If single channel is enabled in multichannel mode, and if frame sync duration is one less than the word length (example word length is 16 bits and frame sync duration is 15 SCLK cycles), then every second frame sync should be taken as invalid frame sync and no data should be transmitted/received for that frame sync. However data is also transmitted/received for premature frame syncs. If this bit is set, no data is transmitted/received for premature frame syncs.
6	COMPANDEN (Applies to ADSP-2147x, ADSP-2148x)	Enable Companding on First Active Channel. If companding for any active channel is enabled in multichannel mode, and the first active channel is not the zeroth channel, and companding is enabled for the first active channel (for example channel 2), then from second frame onwards companding for the first active channel (channel 2) does not occur. If this bit is set, companding does occur for the first active channel, even if it is not the zeroth channel.

SPORT Multichannel Control Registers (SPMCTLx)

The serial ports in the ADSP-214xx processors work individually, not in pairs. Therefore, each SPORT has its own multichannel control register. These registers are shown in [Figure A-90](#) and described in [Table A-88](#).

Note that in ADSP-2136x SHARC processors there is one SPMCTL_{xy} register for each TDM pair, so it's enough to write into one register, for both SPORTs.

On the ADSP-214xx SHARC processors, each sport has its own SPMCTL_x register, so only one write into both SPMCTL_x registers is required to operate the SPORTs as pairs. Since there is no change in SPMCTL_x register bit definitions, the same value can be written into both SPMCTL_x registers in order to make legacy programs for the ADSP-2136x processors operate correctly.

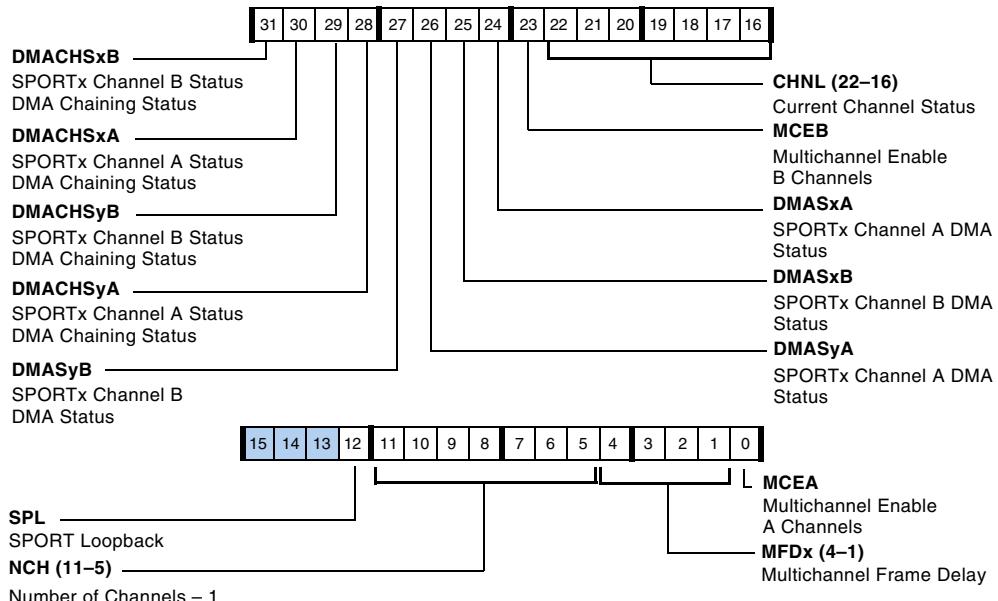


Figure A-90. SPMCTLx Registers – Multichannel Mode

Peripherals Routed Through the DAI

Table A-88. SPMCTLx Register Bit Descriptions (RW)

Bit	Name	Description
0	MCEA	Multichannel Mode Enable, A Channels. Packed and multichannel A modes only. One of two configuration bits that enable and disable multichannel mode on serial port channels. See OPMODE bit (17). 0 = Disable multichannel A operation 1 = Enable multichannel A operation/packed mode Note if MCEA bit is set, the corresponding SPEN_A bit in the SPCTL register should be cleared.
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: $\text{NCH} = \text{Actual number of channel slots} - 1.$
12	SPL	SPORT Loopback Mode. Enables if set (= 1) or disables if cleared (= 0) the channel loopback mode. Loopback mode enables debug capabilities. 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. Only the transmitter acts as master to generate the clock and frame sync. The SPL bit applies to all non multichannel modes. SPORT0 and SPORT1. SPORT0 can only be paired with SPORT1, controlled by the SPL bit in the SPMCTL0/1 registers. SPORT2 and SPORT3. SPORT2 can only be paired with SPORT3, controlled by the SPL bit in the SPMCTL2/3 registers. SPORT4 and SPORT5. SPORT4 can only be paired with SPORT5, controlled by the SPL bit in the SPMCTL4/5 registers. SPORT6 and SPORT7. SPORT6 can only be paired with SPORT7, controlled via SPL bit in the SPMCTL6/7 registers

Table A-88. SPMCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
15–13	Reserved	
22–26 (RO)	CHNL	Current Channel Selected. Identify the currently selected transmit channel slot (0 to 127).
23	MCEB	Multichannel B Mode Enable. Packed and multichannel B modes only. One of two configuration bits that enable and disable multichannel mode on serial port channels. See OPMODE bit (17). 0 = Disable multichannel B operation 1 = Enable multichannel B operation/packed mode Note if MCEB bit is set, the corresponding SPEN_B bit in the SPCTL register should be cleared.
25–24 (RO)	DMASy	DMA y Status. Selects the transfer status. 0 = Inactive 1 = Active
27–26 (RO)	DMASx	DMA x Status. Selects the transfer status. 0 = Inactive 1 = Active
29–28 (RO)	DMACHSy	DMA y Chaining Status. Selects the transfer status. 0 = Inactive 1 = Active
31–30 (RO)	DMACHSx	DMA x Chaining Status. Selects the transfer status. 0 = Inactive 1 = Active

SPORT Active Channel Select Registers (MTxCSy or MRxCSy)

Each bit, 31–0, set (=1) in one of the four MTxCs0, MTxCs1, MTxCs2, MTxCs3 registers for SPORT0/2/4/6 and MRxCs0, MRxCs1, MRxCs2, MRxCs3 for SPORT/1/3/5/7 registers corresponds to the active channel, 127–0, on a multichannel mode serial port. When these registers activate a channel (by setting the respective bits in these registers to 1, the serial port transmits or receives the word in that channel's position of the data stream. When a channel's bit in these registers is cleared (=0), the serial port's data transmit pin three-states during the channel's transmit time slot if the serial

Peripherals Routed Through the DAI

port is configured as transmitter. If the serial port is configured as the receiver it ignores the incoming data.

SPORT Compand Registers (MTxCCSy or MRxCCSy)

Each bit, 31–0, set (=1) in one of the four MTxCCS0, MTxCCS1, MTxCCS2, MTxCCS3 for SPORT/0/2/4/6 and MRxCCS0, MRxCCS1, MRxCCS2, MRxCCS3 for SPORT1/3/5/7 registers corresponds to a companded channel, 127–0, on a multichannel mode serial port. When these registers activate companding for a channel, the SPORT applies the companding from the serial port's DTYPEn selection to the word transmitted or received in that channel's position of the data stream. When a channel's bit in these registers is cleared (=0), the SPORT does not compand the outgoing or incoming data during the channel's time slot.

Error Control Register (SPERRCTLx)

The SPERRCTLx registers control and report the status of the interrupts generated by each SPORT (see [Figure A-91](#), [Table A-89](#)).

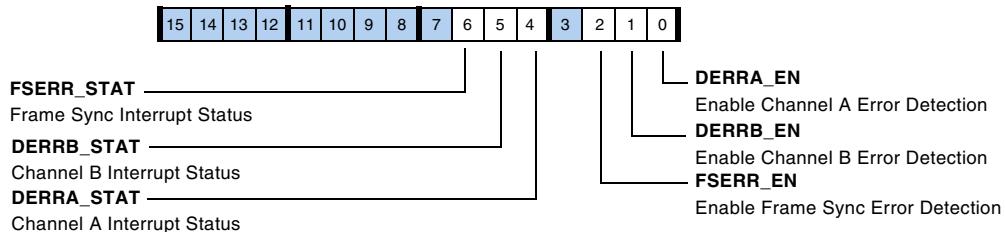


Figure A-91. SPERRCTLx Register

Table A-89. SPERRCTLx Register Bit Descriptions (RW)

Bit	Name	Description
0	DERRA_EN	Enable Channel A Error Detection. 0 = Disable 1 = Enable
1	DERRB_EN	Enable Channel B Error Detection. 0 = Disable 1 = Enable
2	FSERR_EN	Enable Frame Sync Error Detection. 0 = Disable 1 = Enable
3	Reserved	
4 (W1C)	DERRA_STAT	Channel A Interrupt Status. SPTRAN = 0 Receive overflow status SPTRAN = 1 Transmit underflow status
5 (W1C)	DERRB_STAT	Channel B Interrupt Status. SPTRAN = 0 Receive overflow status SPTRAN = 1 Transmit underflow status
6 (W1C)	FSERR_STAT	Frame Sync Interrupt Status. 0 = No frame sync error 1 = Frame sync error detected

Peripherals Routed Through the DAI

SPORT Error Status Register (SPERRSTAT)

The SPERRSTAT register checks the status of SPORT interrupts (see Figure A-92).

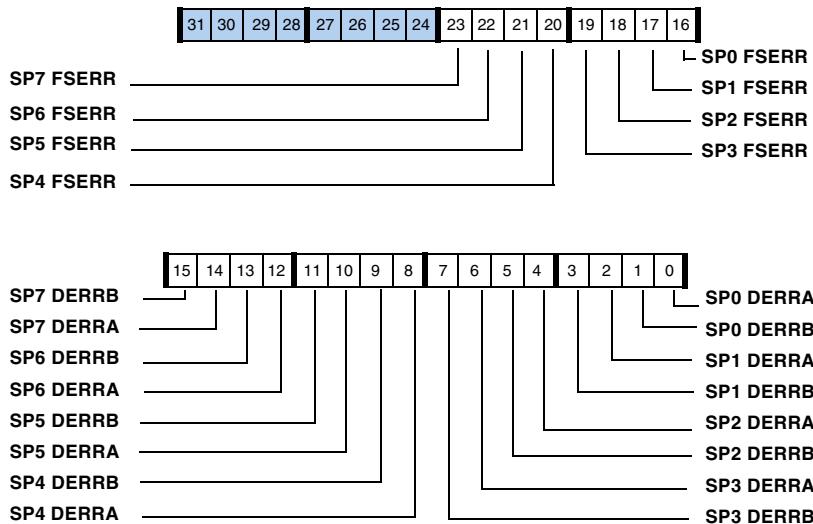


Figure A-92. SPERRSTATx Register (RO)

Input Data Port Registers

The input data port (IDP) provides an additional input path to the processor core. The IDP can be configured as 8 channels of serial data or 7 channels of serial data and a single channel of up to a 20-bit wide parallel data.

Input Data Port DMA Control Registers

For information on these registers, see “[Standard DMA Parameter Registers](#)” on page [2-4](#).

Input Data Port Control Register 0 (IDP_CTL0)

Use this register to configure and enable the IDP and each of its channels. The register is shown in [Figure A-93](#) and described in [Table A-90](#).

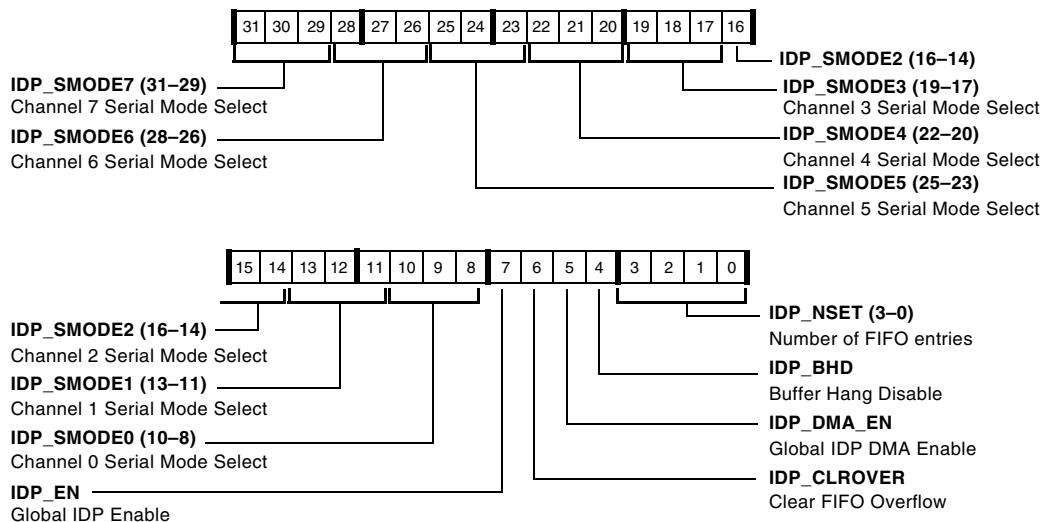


Figure A-93. IDP_CTL0 Register

Peripherals Routed Through the DAI

Table A-90. IDP_CTL0 Register Bit Descriptions (RW)

Bits	Name	Description
3–0	IDP_NSET	<p>Threshold Interrupt. The contents of the IDP_NSET bits represent a threshold number of entries (N) in the FIFO. When the FIFO fills to a point where it has more (N+1) 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 in DAI_IRPTL_x registers.</p> <p>Only the core can use this N threshold interrupt to detect when data needs to be read. The maximum IDP_NSET= 7, otherwise no interrupt is generated.</p>
4	IDP_BHD	<p>IDP Buffer Hang Disable. Reads of an empty FIFO or writes to a full FIFO to cause a core hang. This condition continues until the FIFO has valid data (in the case of reads) or the FIFO has at least one empty location (in the case of writes). This can be used in debug operations.</p> <p>0 = Core hang is enabled 1 = Core hang is disabled</p>
5	IDP_DMA_EN	<p>DMA Enable. Enables DMA on all IDP channels. This bit is the global control for standard and ping-pong DMA.</p> <p>0 = Channel disabled 1 = Channel enabled</p>
6 (WOC)	IDP_CLROVR	FIFO Overflow Clear Bit. Clears the SRU_OVFx bits.
7	IDP_EN	<p>Enable IDP. This bit enables the IDP. This is a global control bit. This bit needs to be set for all operations modes including DMA. When this bit is cleared (= 0), the IDP is disabled, and data cannot move to the IDP_FIFO.</p> <p>Writing a 1 to bit 31 of the IDP_CTL1 register also flushes the FIFO. To enable the IDP for DMA, two separate bits in two different registers must be set. The first are the global IDP_EN and IDP_DMA_EN bits in the IDP_CTL0 register and the second are the specific channel enable bits, located in the IDP_CTL1 register.</p>

Table A-90. IDP_CTL0 Register Bit Descriptions (RW) (Cont'd)

Bits	Name	Description
10–8	IDP_SMODE0	Serial Input Data Format Mode Select. These eight inputs (0–7), each of which contains 3 bits, indicate the mode of the serial input for each of the eight IDP channels.
13–11	IDP_SMODE1	
16–14	IDP_SMODE2	Input format: 000 = Left-justified 24 bits 001 = I ² S mode 24 bits 010 = Left-justified 32 bits
19–17	IDP_SMODE3	011 = I ² S 32 bits
22–20	IDP_SMODE4	100 = Right-justified 24 bits 101 = Right-justified 20 bits
25–23	IDP_SMODE5	110 = Right-justified 18 bits
28–26	IDP_SMODE6	111 = Right-justified 16 bits
31–29	IDP_SMODE7	Note the SMODEx bits define the IDP buffer input format for core access. For I2S and left-justified single channel modes, it receives 32 bits of data from the SDATA pins. No L/R bit is added in these modes.

Input Data Port Control Register 1 (IDP_CTL1)

Use the IDP_CTL1 register to configure and enable individual IDP channels. The register is shown in [Figure A-94](#) and described in [Table A-91](#).

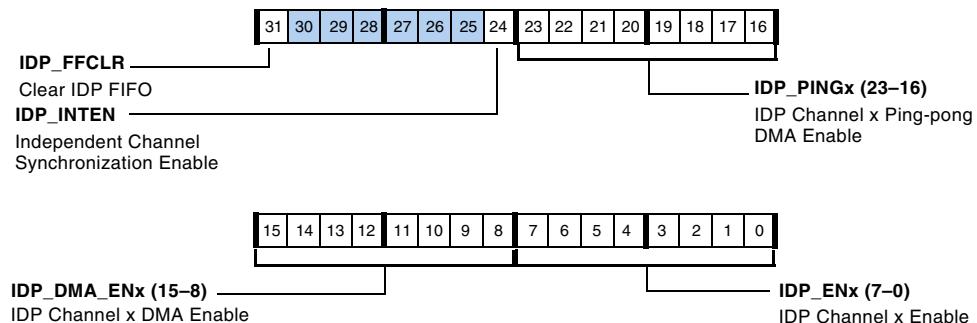


Figure A-94. IDP_CTL1 Register

Peripherals Routed Through the DAI

Table A-91. IDP_CTL1 Register Bit Descriptions (RW)

Bit	Name	Description
7–0	IDP_ENx	IDP Channel x Enable. These are the enable bits for accepting data from individual channels. Corresponding IDP_ENx must be set with IDP_EN bit to get data from channel x. If IDP_EN bit is not set then this bit has no effect. 0x00 = all channels cleared 0xFF = all channels enabled (default)
15–8	IDP_DMA_ENx	IDP DMA Enable. These are the DMA enable bits for individual channels. Corresponding IDP_DMA_ENx must be set with IDP_DMA_EN bit for DMA transfer of data from channel x. If the global DMA_EN bit is not set then this bit has no effect. 0x00 = all channels cleared 0xFF = all channels enabled (default)
23–16	IDP_PINGx	IDP Ping-Pong DMA Channel x Enable. These are the Ping-Pong DMA enable bits for individual channels. Corresponding IDP_PINGx must be set to start ping-pong DMA from channel x. This bit requires the IDP_DMA_ENx bit and IDP_DMA_EN bit are set.
24	IDP_INTEN	Independent Channel Synchronization Enable. This is the enable bit for independent channel synchronization. If this bit is set, the IDP channels will start shifting in data from the first active edge of the LRCLK based on the setting of FAEx. If this bit is cleared (reset value), then the ADSP-214xx behaves like previous SHARC processors.
30–25	Reserved	
31 (WOC)	IDP_FFCLR	Clear IDP FIFO. Setting this bit to 1 clears the IDP FIFO and the IDP_FIFOSZ bits. Note that when the IDP_EN bit transitions from 1 to 0, all data in the IDP FIFO are flushed.

Input Data Port Control Register 2 (IDP_CTL2)

This register controls the first active edge selection for channel synchronization (and is only available on the ADSP-2147x and ADSP-2148x processors). The register is shown in [Figure A-95](#) and described in [Table A-92](#).

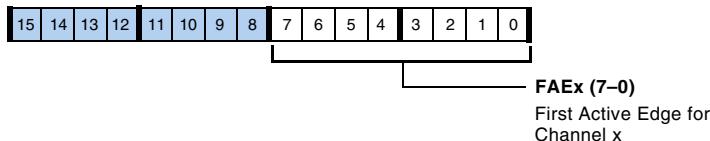


Figure A-95. IDP_CTL2 Register

Table A-92. IDP_CTL2 Register Bit Descriptions (RW)

Bit	Name	Description
7–0	FAEx	First Active Edge for Channel x. 1 = nth IDP channel starts shifting in data from the first rising edge of LRCLK after IDP is enabled. This data is latched after the next falling edge of LRCLK. 0 = nth IDP channel starts shifting in data from the first falling edge of LRCLK after IDP is enabled. This data is latched after the next rising edge of LRCLK. Reset value of all these bits is 0. These bits are used only if IDP_INTEN bit (IDP_CTL1[24]) is set.
8–31	Reserved	

Parallel Data Acquisition Port Control Register (IDP_PP_CTL)

The IDP_PP_CTL register (shown in [Figure A-96](#) and described in [Table A-93](#)) provides 20 mask bits that allow the input from any of the 20 pins to be ignored.

For more information on the operation of the parallel data acquisition port, see [Chapter 11, Input Data Port](#). For information on the pin muxing that is used in conjunction with this module, see “[Pin Multiplexing](#)” on page [23-28](#).

Peripherals Routed Through the DAI

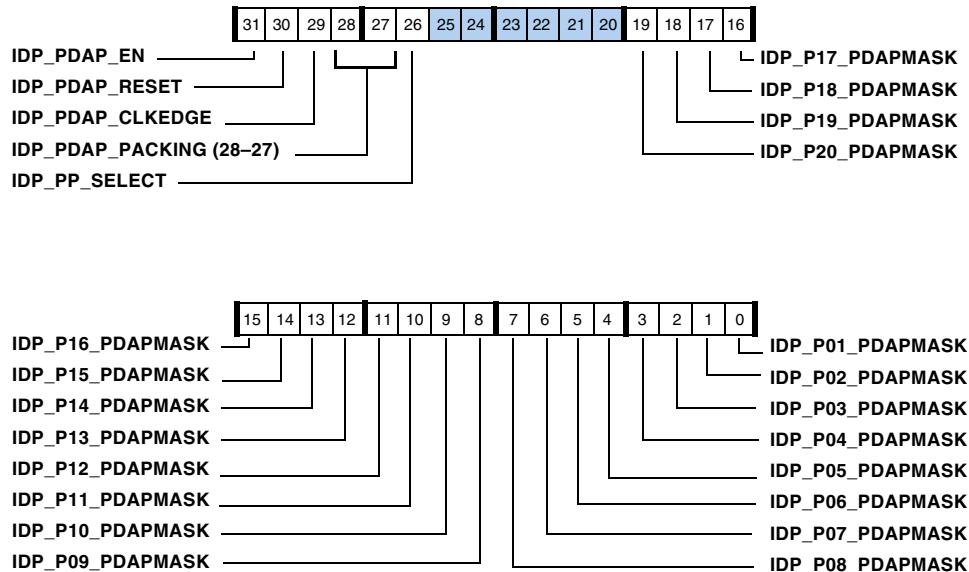


Figure A-96. IDP_PP_CTL Register

Table A-93. IDP_PP_CTL Register Bit Descriptions (RW)

Bit	Name	Description
19–0	IDP_P20_1_PDAPMASK	Parallel Data Acquisition Port Mask. For each of the parallel inputs: 0 = Input data from PDAP_20-1 are masked 1 = Input data from PDAP_20-1 are unmasked After this masking process, data gets passed along to the packing unit.
25–20	Reserved	
26	IDP_PP_SELECT	PDAP Port Select. This bit selects which peripheral is connected to the PDAP unit. 0 = Data/control bits are read from DAI pins 1 = Data/control bits are read from AMI_ADDR pins

Table A-93. IDP_PP_CTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
28–27	IDP_PDAP_PACKING	<p>Packing. Selects PDAP packing mode. These bits mask parallel sub words from the 20 parallel input signals and packs them into a 32-bit word. The bit field indicates how data is packed. Selection of packing mode is made based on the application.</p> <p>00 = 8- to 32-bit (packing by 4) 01 = (11, 11, 10) to 32-bit (packing by 3) 10 = 16- to 32-bit (packing by 2) 11 = 20- to 32-bit (no packing). 12 LSBs are cleared Note for input data width less than 20-bits, inputs are aligned to MSB pins.</p>
29	IDP_PDAP_CLKEDGE	<p>PDAP (Rising or Falling) Clock Edge. Setting this bit (= 1) causes the data to latch on the falling edge (PDAP_CLK_I signal). Clearing this bit (= 0) causes data to latch on the rising edge (default).</p> <p>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.</p> <p>0 = Data is latched on the rising edge 1 = Data is latched on the falling edge</p>
30 (WO)	IDP_PDAP_RESET	PDAP Reset. A reset clears any data in the packing unit waiting to get latched into the IDP FIFO. This bit reset the counter of the PDAP and is useful for packing alignment. This bit always returns a value of zero when read.
31	IDP_PDAP_EN	<p>PDAP Enable. 0 = Disconnects all PDAP inputs (data/control) from use as parallel input channel 1 = Connects all PDAP inputs (data/control) from use as parallel input channel. IDP channel 0 cannot be used as a serial input port with this setting</p>

Peripherals Routed Through the DAI

IDP Status Register (DAI_STAT0)

The IDP DMA status register shown in [Figure A-97](#) and described in [Table A-94](#) reflects the status of the standard and ping-pong DMA channels.

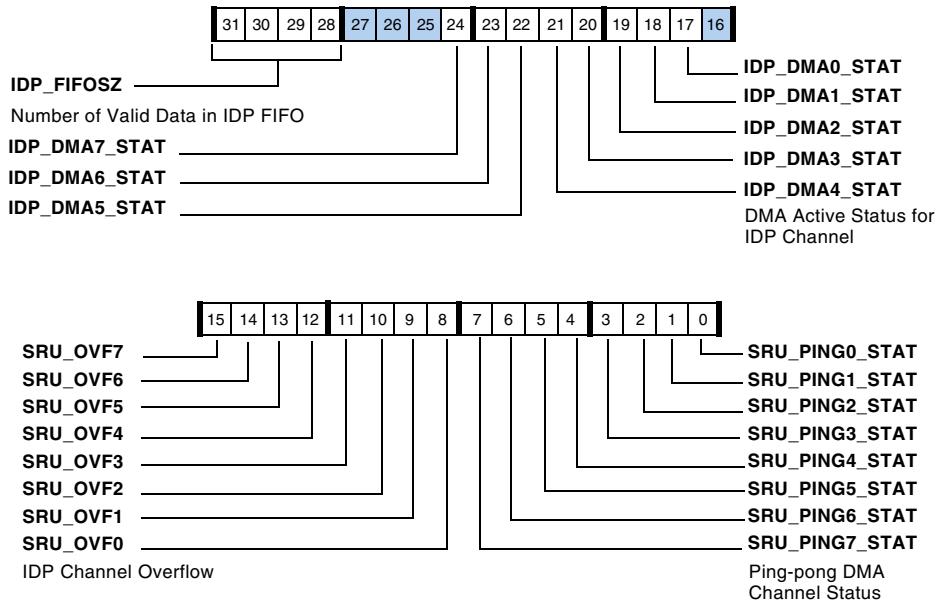


Figure A-97. DAI_STAT0 Register

Table A-94. DAI_STAT0 Register Bit Descriptions (RO)

Bit	Name	Description
7-0	SRU_PINGx_STAT	Ping-Pong DMA Channel Status. Indicates the status of ping-pong DMA in each respective channel (channel 7-0). 0 = DMA is not active 1 = DMA is active

Table A-94. DAI_STAT0 Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
15–8 (Sticky)	SRU_OVFx	Sticky Overflow Channel Status. Provides overflow status information for each channel (bit 8 for channel 0 through bit 15 for channel 7). 0 = SRU input no overflow 1 = SRU input overflow has occurred
16	Reserved	
24–17	IDP_DMAx_ST AT	Input Data Port DMA Channel Status. These bits reflect the state of all eight DMA channels. These bits are set once IDP_DMA_EN is set and remain set until the last data of that channel is transferred. Even if IDP_DMA_EN is set (=1), this bit goes low once the required number of data transfers occur. Note if DMA through some channel is not intended, this bit goes high. 0 = DMA is not active 1 = DMA is active
27–25	Reserved	
31–28	IDP_FIFOSZ	IDP FIFO Size. Indicates Number of samples in the IDP FIFO. 0000 = IDP FIFO empty 1000 = IDP FIFO full

IDP Status Register 1 (DAI_STAT1)

Since the core allows writes to the IDP_FIFO, the DAI_STAT1 register stores the different read or writes indexes with a maximum of 8 entries each.

Table A-95. DAI_STAT1 Register Bit Descriptions (RO)

Bit	Name	Description
3–0	FIFO_WRI	Write Index Pointer. Reflects the write index status during core writes to the IDP_FIFO. 0000 = No write done 1000 = 8 writes done

Peripherals Routed Through the DAI

Table A-95. DAI_STAT1 Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
7–4	FIFO_RDI	Read Index Pointer. Reflects the read index status during core reads from the IDP_FIFO. 0000 = No read done 1000 = 8 reads done
31–8	Reserved	

Sample Rate Converter Registers

The sample rate converter (SRC) is composed of five registers which are described in the following sections.

Control Registers (SRCCTL x)

The SRCCTL n control registers (read/write) control the operating modes, filters, and data formats used in the sample rate converter. For $n = 0$, the register controls the SRC0 and SRC1 modules and for $n = 1$ it controls the SRC2 and SRC3 modules ($x = 0, 2$ and $y = 1, 3$). The bit settings for these registers are shown in [Figure A-98](#) and described in [Table A-96](#).

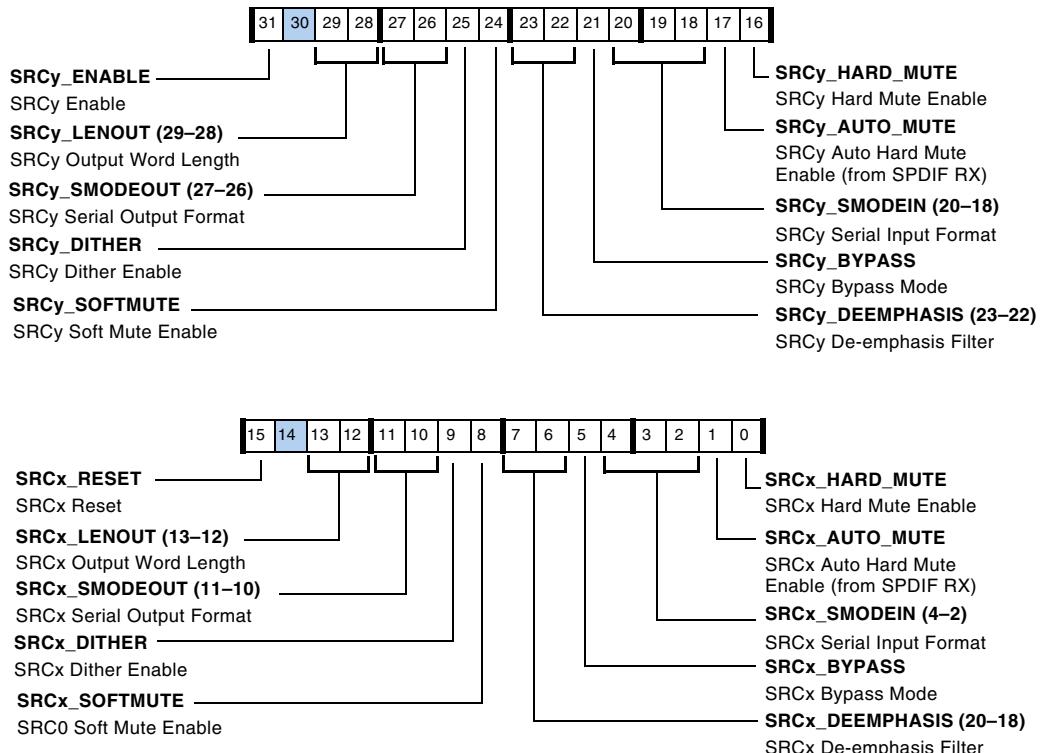


Figure A-98. SRCCTLx Register

Table A-96. SRCCTLx Register Bit Descriptions (RW)

Bit	Name	Description
0	SRCx_HARD_MUTE	Hard Mute. Hard mutes SRC 0, 2. 1 = Mute (default)
1	SRCx_AUTO_MUTE	Auto Hard Mute. Auto hard mutes SRC 0, 2 when non audio is asserted by the SPDIF receiver. 0 = No mute 1 = Mute (default)

Peripherals Routed Through the DAI

Table A-96. SRCCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
2–4	SRCx_SMODEIN	Serial Input Format. Selects the serial input format for SRC 0, 2 as follows: 000 = Default, format is left-justified 001 = I 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	SRCx_BYPASS	Bypass SRCx. Output of SRC 0, 2 is the same as the input.
6–7	SRCx_DEEMPHASIS	De-emphasis Filter Select. Used to de-emphasize audio data that has been emphasized. The type of de-emphasis filter is selected by the SRCx_DEEMPHASIS bits and is based on the input sample rate (SRCx_FS_IP_I signal) as follows: enables de-emphasis on incoming audio data for SRC 0, 2. 00 = No de-emphasis 01 = 32 kHz 10 = 44.1 kHz 11 = 48 kHz
8	SRCx_SOFTMUTE	Soft Mute. Enables soft mute on SRC 0, 2. 0 = No mute 1 = Mute (default)
9	SRCx_DITHER	Dither Select. Enables dithering on SRC 0, 2 when a word length less than 24 bits is selected. 0 = Dithering is enabled (default) 1 = Dithering is disabled
10–11	SRCx_SMODEOUT	Serial Output Format. Selects the serial output format on SRC 0, 2 as follows: 00 = Left-justified (default) 01 = I ² S 10 = TDM mode 11 = Right-justified. The right-justified serial data out mode assumes 64 SCLK cycles per frame, divided evenly for left and right. For the other modes these 8 LSBs contain zeros.

Table A-96. SRCCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
12–13	SRCx_LENOUT	Output Word Length Select. Selects the serial output word length on SRC 0, 2 as follows: 00 = 24 bits 01 = 20 bits 10 = 18 bits 11 = 16 bits Any word length less than 24 bits has dither added to the unused LSBs if SRCx_DITHER is enabled (= 1).
14	Reserved	
15	SRCx_ENABLE	SRC0 Enable. Enables SRC 0, 2. When (set = 1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine where; 1) MUTE_OUT is asserted, 2) soft mute control counter for input samples is set to maximum attenuation (-144 dB). Note that SRC power-up completion is finished by clearing the SRCx_MUTEOUT bit in SRCRATx register. Writes to the SRCCTLx register should be at least one cycle before setting the SRCx_ENABLE. When setting and clearing this bit, it should be held low for a minimum of 5 PCLK cycles. Programs should disable the SRC when changing modes.
16	SRCy_HARD_MUTE	Hard Mute. Hard mutes SRC 1, 3. 1 = Mute (default)
17	SRCy_AUTO_MUTE	Auto Hard Mute. Auto hard mutes SRC 1, 3 when non audio is asserted by the SPDIF receiver. 0 = No mute 1 = Mute (default)
18–20	SRCy_SMODEIN	Serial Input Format. Selects the serial input format for SRC 1, 3 as follows: 000 = Default, format is left-justified 001 = I 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	SRCy_BYPASS	Bypass Mode Enable. Output of SRC 1, 3 is the same as input.

Peripherals Routed Through the DAI

Table A-96. SRCCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
22–23	SRCy_DEEMPHASIS	De-emphasis Filter Select. Enables de-emphasis on incoming audio data for SRC 1, 3. 00 = No de-emphasis 01 = 32 kHz 10 = 44.1 kHz 11 = 48 kHz
24	SRCy_SOFTMUTE	Soft Mute. Enables soft mute on SRC 1, 3. 0 = No mute 1 = Mute (default)
25	SRCy_DITHER	Dither Select. Disables dithering on SRC 1, 3 when a word length less than 24 bits is selected. 0 = Dithering is disabled (default) 1 = Dithering is enabled
26–27	SRCy_SMODEOUT	Serial Output Format. Selects the serial output format for SRC 1, 3 as follows. 00 = Left-justified (default) 01 = I ² S 10 = TDM mode 11 = Right-justified
28–29	SRCy_LENOUT	Output Word Length Select. Selects the serial output word length for SRC 1, 3 as follows: 00 = 24 bits 01 = 20 bits 10 = 18 bits 11 = 16 bits Any word length less than 24 bits has dither added to the unused LSBs if SRCx_DITHER is enabled (= 0).

Table A-96. SRCCTLx Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
30	Reserved	
31	SRCy_ENABLE	<p>SRC Enable. Enables SRC 1, 3. When (set = 1), or when the sample rate (frame sync) between the input and output changes, the SRC begins its initialization routine where;</p> <ol style="list-style-type: none"> 1) MUTE_OUT is asserted, 2) soft mute control counter for input samples is set to maximum attenuation (-144 dB). <p>Note that SRC power-up completion is finished by clearing the SRCx_MUTEOUT bit in SRCRATx register. Writes to the SRCCTLx register should be at least one cycle before setting the SRCx_ENABLE. When setting and clearing this bit, it should be held low for a minimum of 5 PCLK cycles. Programs should disable the SRC when changing modes.</p>

Mute Register (SRCMUTE)

This register connects an SRCx mute input and output when the SRC0_MUTE_ENx bit is cleared (=0). This allows SRCx to automatically mute input while the SRC is initializing (0 = automatic muting and 1 = manual muting). Bit 0 controls SRC0, bit 1 controls SRC1, bit 2 controls SRC2, and bit 3 controls SRC3.

Ratio Registers (SRCRATx)

These registers report the mute and I/O sample ratio as follows: the SRCRAT0 register reports for SRC0 and SRC1 and the SRCRAT1 register reports the mute and I/O sample ratio for SRC2 and SRC3 (X = SRC0 and SRC1, Y = SRC2 and SRC3). The registers are shown in [Figure A-99](#) and [Figure A-100](#) and described in [Table A-97](#).

Peripherals Routed Through the DAI

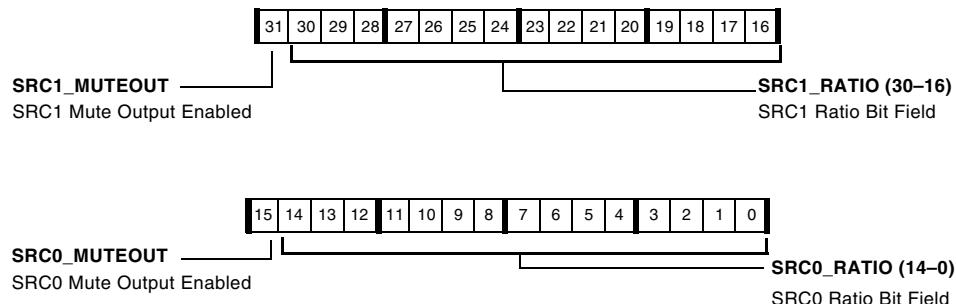


Figure A-99. SRCRAT0 Register

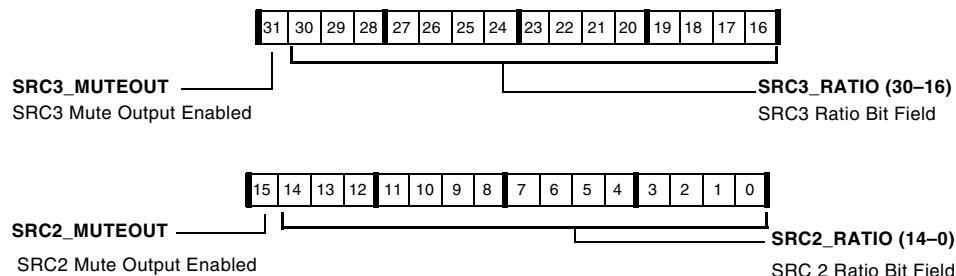


Figure A-100. SRC SRCRAT1 Register

Table A-97. SRCRATx Register Bit Descriptions (RO)

Bit	Name	Description
14–0	SRCx_RATIO	Sampling Ratio of Frame Syncs. These bits can be read to find the ratio of output to input sampling frequency ($\text{SRCx_FS_OP_I}/\text{SRCx_FS_IP_I}$). This ratio is reported in 4.11 (integer.fraction) format where the 15-bit value of the normal binary number is comprised of 4 bits for the integer and 11 bits for the fraction.
15	SRCx_MUTEOUT	Mute Status. The SRCx_MUTEOUT bits in SRCRATx register report the status of the MUTE_OUT signal. Once the SRCx_MUTEOUT signal is cleared then the ratio can be read. When the SRCx_ENABLE is set or there is a change in the sample ratio, the MUTE_OUT signal is asserted. The MUTE_OUT signal remains asserted until the digital servo loop's internal fast settling mode is complete. When the digital servo loop has switched to slow settling mode, the MUTE_OUT signal is deasserted. Reset = 0x80008000.
30–16	SRCy_RATIO	Sampling Ratio of Frame Syncs. See bits 14–0.
31	SRCy_MUTEOUT	Mute Status. See bit 15.

Precision Clock Generator Registers

The precision clock generator (PCG) consists of four identical units. Each of these units (A, B, C, and D) generates one clock (`CLKA_0` `CLKB_0`, `CLKC_0` or `CLKD_0`) and one frame sync (`FSA_0`, `FSB_0`, `FSC_0` or `FSD_0`) output.

Control Registers (PCG_CTLxy)

Two control registers ($y=0, 1$) operate for each unit. The control registers enable clocks, frame syncs, and select divisors for each unit. These registers are shown in [Figure A-101](#) and [Figure A-102](#) and described in [Table A-98](#) and [Table A-99](#). Note the different units ($x = A, B, C, D$) any clock unit can be chosen.

Peripherals Routed Through the DAI

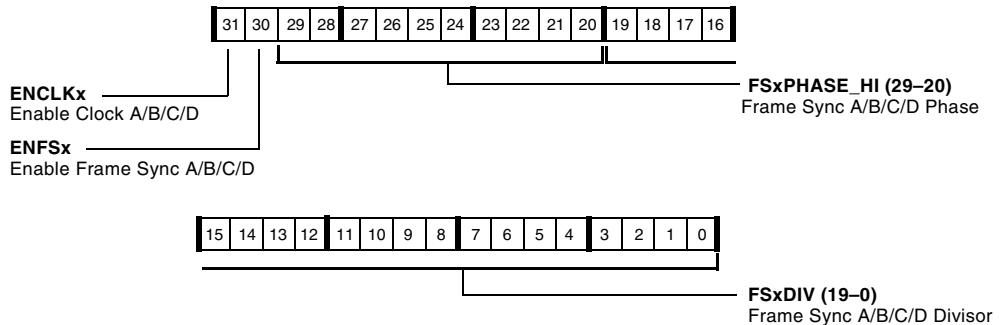


Figure A-101. PCG_CTLx0 Registers

Table A-98. PCG_CTLx0 Register Bit Descriptions (RW)

Bit	Name	Description
19–0	FSxDIV	Divisor for Frame Sync A/B/C/D. This 20-bit field frame sync divider is multiplexed: FSxDIV >1 PCGx is in normal mode FSxDIV =0,1 PCGx is in bypass mode For more information on bypass mode, refer to the STROBEx and INVFSx bits of the PCG_PWx register.
29–20	FSxPHASE_HI	Phase for Frame Sync A/B/C/D. This field represents the upper half of the 20-bit value for the channel A/B/C/D frame sync phase. See also FSXPHASE_LO (Bits 29–20) in Table A-99 .
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

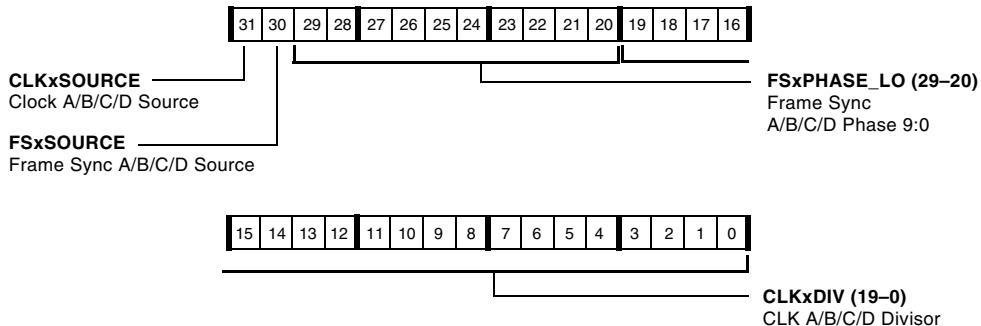


Figure A-102. PCG_CTLx1 Register

Table A-99. PCG_CTLx1 Register Bit Descriptions (RW)

Bit	Name	Description
19–0	CLKxDIV	Divisor for Clock A/B/C/D.
29–20	FSxPHASE_LO	Phase for Frame Sync A/B/C/D. This field represents the lower half of the 20-bit value for the channel A/B/C/D frame sync phase. See also FSXPHASE_HI (Bits 29-20) in PCG_CTLx1 described on on page A-192 .
30	FSxSOURCE	Frame Sync Source. Master clock source for frame sync A/B/C/D. 0 = CLKIN pin selected for specified frame sync 1 = PCG_EXTx_I selected for specified frame sync This frame sync period is also a reference for the strobe period in one shot mode.
31	CLKxSOURCE	Clock Source. Master clock source for clock A/B/C/D. 0 = CLKIN pin selected for specified clock 1 = PCG_EXTx_I selected for specified clock

Clock Inputs

The **CLKxSOURCE** bit (bit 31 in the **PCG_CTLx1** registers) specifies the input source for the clock of the respective units (A, B, C, and D). When this bit is cleared (= 0), the input is sourced from the external oscillator/crystal, as shown in [Figure 14-1 on page 14-6](#). When set (= 1), the input is sourced

Peripherals Routed Through the DAI

from DAI. The CLKxSOURCE bit is overridden if CLKx_SOURCE_IOP bit in the PCG_SYNCx register is set. If the CLKx_SOURCE_IOP bit is set, the input is sourced from the peripheral clock (PCLK).

Pulse Width Registers (PCG_PWx)

Pulse width is the number of input clock periods for which the frame sync output is HIGH. Pulse width should be less than the divisor of the frame sync. The pulse width control registers are shown in [Figure A-103](#) and [Figure A-104](#) and described in [Table A-100](#) and [Table A-101](#). Note that where letters and slashes appear, for example A/B/C/D, any clock unit can be chosen.

If the STROBEA/B/C/D bits of the pulse width control register (PCG_PW, PCG_PW2) is reset to 0, then the input is directly passed to the frame sync output, either not inverted or inverted, depending on the INVFA, INVFB, INVFC and INVFD bits of the PCG_PW and PCG_PW2 registers.

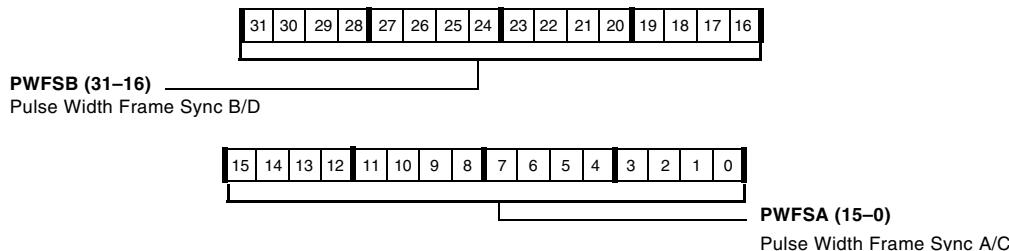


Figure A-103. PCG_PWx Registers (in Normal Mode)

Table A-100. PCG_PWx Register Bit Descriptions (in Normal Mode)
(RW)

Bit	Name	Description
15–0	PWFSA	Pulse Width for Frame Sync A/C. Note: This is valid when not in bypass mode
31–16	PWFSB	Pulse Width for Frame Sync B/D. Note: This is valid when not in bypass mode

In bypass mode, if the least significant bit (LSB) of the PCG_PW register is set to 1, then a one-shot pulse is generated. This one-shot-pulse has a duration equal to the period of MISCA2_I for unit A, MISCA3_I for unit B, MISCA4_I for unit C, and MISCA5_I for unit D (see “[DAI Routing Capabilities](#)” on page 9-24). 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.

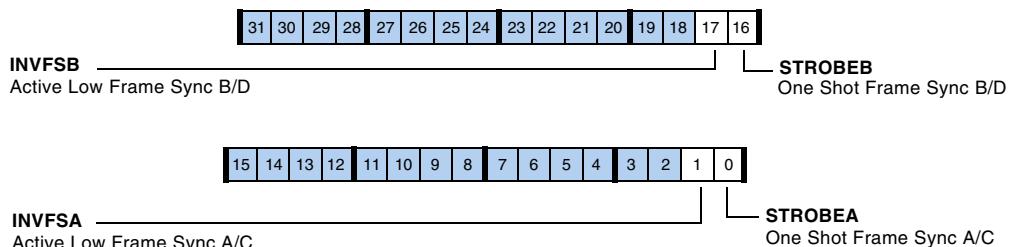


Figure A-104. PCG_PWx Registers (in Bypass Mode)

Table A-101. PCG_PWx Register Bit Descriptions
(in Bypass Mode) (RW)

Bit	Name	Description
0	STROBEx	One Shot Frame Sync A/C. Frame sync is a pulse with duration equal to one period of the MISCA2_I signal (PCG A) MISCA4_I signal (PCG C) repeating at the beginning of every frame. Note: This is valid in bypass mode only.
1	INVFSx	Active Low Frame Sync Select for Frame Sync A/C. 0 = Active high frame sync 1 = Active low frame sync
15–2	Reserved	(In bypass mode, bits 15-2 are ignored.)
16	STROBEx	One Shot Frame Sync B/D. Frame sync is a pulse with duration equal to one period of the MISCA3_I signal (PCG B) MISCA5_I signal (PCG D) repeating at the beginning of every frame. Note: This is valid in bypass mode only.

Peripherals Routed Through the DAI

Table A-101. PCG_PWx Register Bit Descriptions
(in Bypass Mode) (RW) (Cont'd)

Bit	Name	Description
17	INVFSx	Active Low Frame Sync Select. 0 = Active high frame sync 1 = Active low frame sync
31–18	Reserved (In bypass mode, bits 31–18 are ignored.)	

PCG Frame Synchronization Registers (PCG_SYNCx)

These registers, shown in [Figure A-105](#), and [Figure A-106](#) and described in [Table A-102](#) and [Table A-103](#), allow programs to synchronize the clock frame sync units with external frame syncs.

Note the CLKxSOURCE bits (PCG_CTLx1 register) are overridden if CLKx_SOURCE_IOP bits (bit 2) in the PCG_SYNCx registers are set.

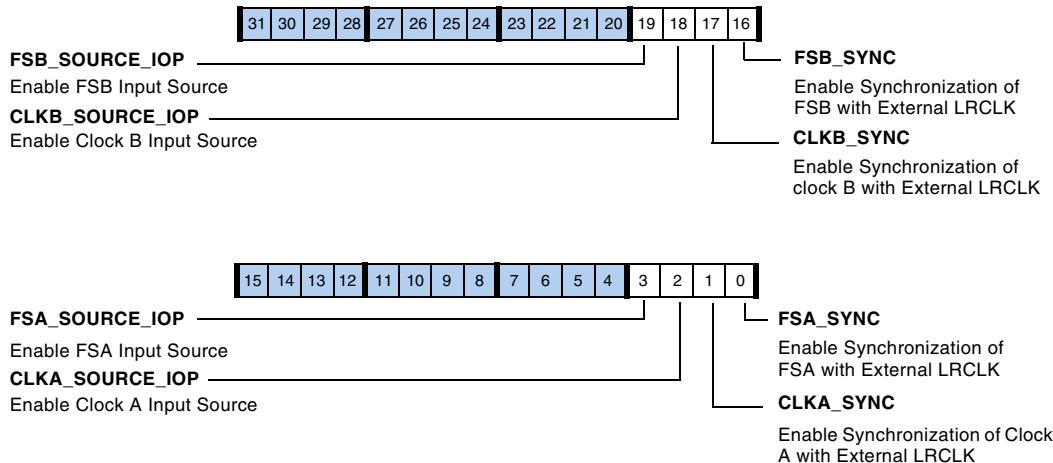


Figure A-105. PCG_SYNC1 Register

Table A-102. PCG_SYNC1 Register Bit Descriptions (RW)

Bit	Name	Description
0	FSA_SYNC	Enable Synchronization of Frame Sync A With External Frame Sync. 0 = Frame sync disabled 1 = Frame sync enabled
1	CLKA_SYNC	Enable Synchronization of Clock A With External Frame Sync. 0 = Clock disabled 1 = Clock enabled
2	CLKA_SOURCE_IOP	Enable Clock A Input Source. 0 = Output selected by CLKASOURCE bit 1 = PCLK selected for clock A.
3	FSA_SOURCE_IOP	Enable Frame Sync A Input Source. 0 = Output selected by FSASOURCE bit 1 = PCLK selected for frame sync A.
16	FSB_SYNC	Enable Synchronization of Frame Sync B With External Frame Sync. 0 = Frame sync disabled 1 = Frame sync enabled
17	CLKB_SYNC	Enable Synchronization of Clock B With External Frame Sync. 0 = Clock disabled 1 = Clock enabled
18	CLKB_SOURCE_IOP	Enable Clock B Input Source. 0 = Output selected by CLKBSOURCE bit 1 = PCLK selected for clock B.
19	FSB_SOURCE_IOP	Enable Frame Sync B Input Source. 0 = Output selected by FSBSOURCE bit 1 = PCLK selected for frame sync B.

Peripherals Routed Through the DAI

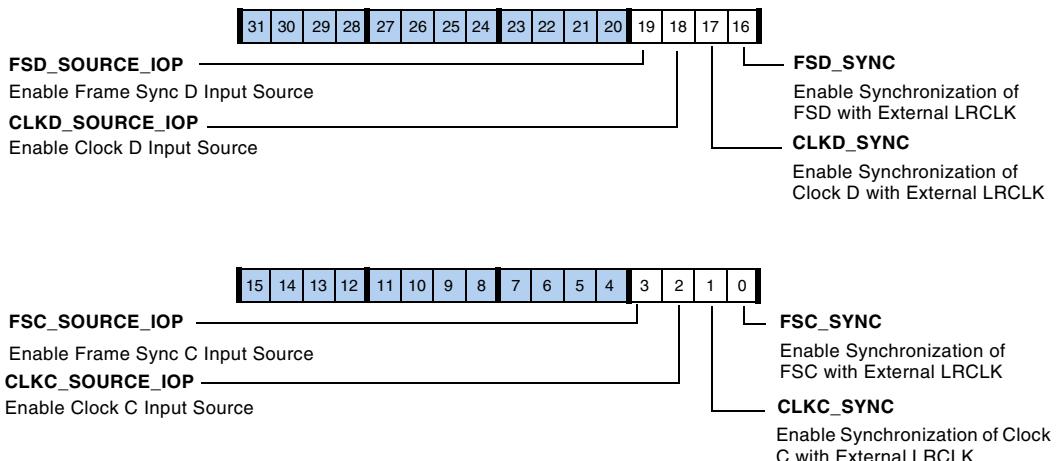


Figure A-106. PCG_SYNC2 Register

Table A-103. PCG_SYNC2 Register Bit Descriptions (RW)

Bit	Name	Description
0	FSC_SYNC	Enable Synchronization of Frame Sync C With External Frame Sync. 0 = Frame sync disabled 1 = Frame sync enabled
1	CLKC_SYNC	Enable Synchronization of Clock C With External Frame Sync. 0 = Clock disabled 1 = Clock enabled
2	CLKC_SOURCE_IOP	Enable Clock C Input Source. 0 = Output selected by CLKCSOURCE bit 1 = PCLK selected for clock C
3	FSC_SOURCE_IOP	Enable Frame Sync C Input Source. 0 = Output selected by FSCSOURCE bit 1 = PCLK selected for frame sync C
16	FSD_SYNC	Enable Synchronization of Frame Sync D With External Frame Sync. 0 = Frame sync disabled 1 = Frame sync enabled

Table A-103. PCG_SYNC2 Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
17	CLKD_SYNC	Enable Synchronization of Clock D With External Frame Sync. 0 = Clock disabled 1 = Clock enabled
18	CLKD_SOURCE_IOP	Enable Clock D Input Source. 0 = Output selected by CLKDSOURCE bit 1 = PCLK selected for clock D
19	FSD_SOURCE_IOP	Enable Frame Sync D Input Source. 0 = Output selected by FSDSOURCE bit 1 = PCLK selected for frame sync D

Sony/Philips Digital Interface Registers

The following sections describe the registers that are used to configure, enable, and report status information for the S/PDIF transceiver.

Transmitter Registers

The following sections describe the S/PDIF transmitter registers.

Transmit Control Register (DITCTL)

This 32-bit register's bits are shown in [Figure A-107](#) and described in [Table A-104](#).

Peripherals Routed Through the DAI

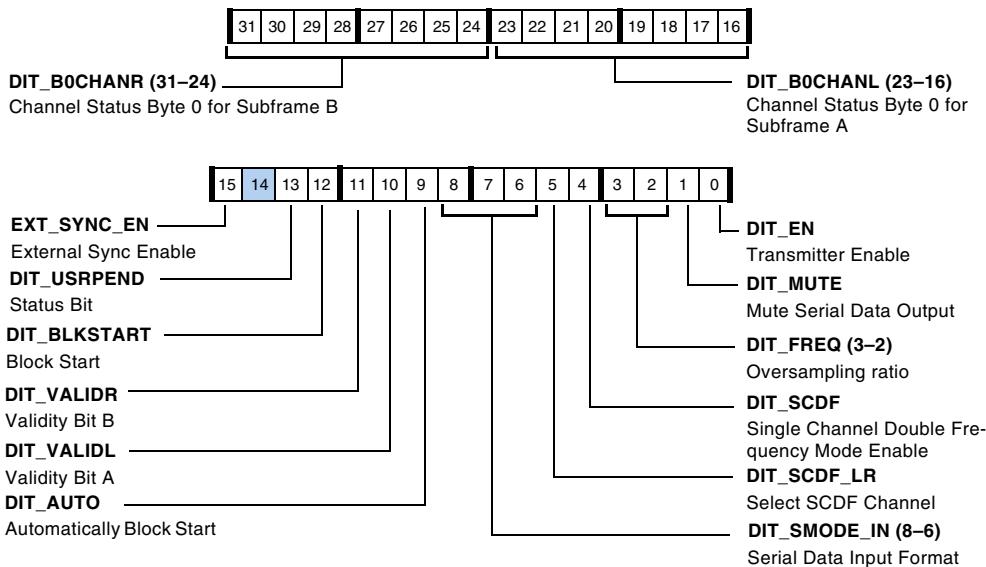


Figure A-107. DITCTL Register

Table A-104. DITCTL Register Bit Descriptions (RW)

Bit	Name	Description
0	DIT_EN	Transmitter Enable. Enables the transmitter and resets the control registers to their defaults. 0 = Transmitter disabled 1 = Transmitter enabled
1	DIT_MUTE	Mute. Mutes the serial data output.
3–2	DIT_FREQ	Frequency Multiplier. Sets the over sampling ratio to the following: 00 = 256 × frame sync 01 = 384 × frame sync
4	DIT_SCDF	Single-Channel, Double-Frequency Mode Enable. 0 = 2 channel mode 1 = SCDF mode
5	DIT_SCDF_LR	Select Single-Channel, Double-Frequency Mode. 0 = Left channel 1 = Right channel

Table A-104. DITCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
8–6	DIT_SMODEIN	Serial Data Input Format. Selects the input format as follows: 000 = Left-justified 001 = I 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_AUTO	Automatically Generate Block Start. Automatically generate block start. When enabled, the transmitter is in standalone mode where it inserts block start, channel status, and validity bits on its own. If the channel status or validity buffer needs to be enabled (after the SRU programming is complete), first write to the buffers with the required data and then enable the buffers by setting the DIT_AUTO bit. 0 = Manually start block transfer according to input stream status bits 1 = Automatically start block transfer.
10	DIT_VALIDL	Validity Bit A. Use with channel status buffer.
11	DIT_VALIDR	Validity Bit B. Use with channel status buffer.
12 (RO)	DIT_BLKSTART	Block Start. Status bit that indicates block start (when bit 9, DIT_AUTO = 1). 0 = Current word is not block start 1 = Current word is block start
13 (RO)	DIT_USRPEND	User Bits Pending. This bit is set if the update of the internal buffer from the DITUSRBITA/Bx registers has not completed yet.
14	Reserved	
15	EXT_SYNC_EN	External Sync Enable. When set (Regardless of bit 9) the internal frame counter is set to zero at an internal LRCLK rising edge followed by an DIT_EXTSYNC_I rising edge.
23–16	DIT_B0CHANL	Channel Status Byte 0 for Subframe A.
31–24	DIT_B0CHANR	Channel Status Byte 0 for Subframe B.

Peripherals Routed Through the DAI

Transmit Status Bit Registers for Subframe A/B (DITCHANAx/Bx)

These registers provide status information for transmitter subframe A and B. The first five bytes of the channel status may be written all at once to the control registers for both A and B channels. As the data is serialized and transmitted, the appropriate bit is inserted into the channel status area of the 192-word frame. Note that these registers are used in standalone mode only.

There are six channel status registers associated with subframe A (left channel) and six user bits buffer registers associated with subframe B (right channel). Since a block owns 2 x 192 frames, 24 bytes per frame are required for storage. Note that status byte 0 is available in the DITCTL register. These registers are listed with their locations in [Table A-105](#) and [Table A-106](#).

Table A-105. DITCHANAx Registers (RW)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITCTL			BYTE0	
DITCHANA0	BYTE1	BYTE2	BYTE3	BYTE4
DITCHANA1	BYTE5	BYTE6	BYTE7	BYTE8
DITCHANA2	BYTE9	BYTE10	BYTE11	BYTE12
DITCHANA3	BYTE13	BYTE14	BYTE15	BYTE16
DITCHANA4	BYTE17	BYTE18	BYTE19	BYTE20
DITCHANA5	BYTE21	BYTE22	BYTE23	Reserved

Table A-106. DITCHANBx Registers (RW)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITCTL				BYTE0
DITCHANB0	BYTE1	BYTE2	BYTE3	BYTE4
DITCHANB1	BYTE5	BYTE6	BYTE7	BYTE8

Table A-106. DITCHANBx Registers (RW) (Cont'd)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITCHANB2	BYTE9	BYTE10	BYTE11	BYTE12
DITCHANB3	BYTE13	BYTE14	BYTE15	BYTE16
DITCHANB4	BYTE17	BYTE18	BYTE19	BYTE20
DITCHANB5	BYTE21	BYTE22	BYTE23	Reserved

Transmit User Bits Buffer Registers for Subframe A/B Registers (DITUSRBITAx/Bx)

Once programmed, these registers are used only for the next block of data. This allows programs to change the user bit information with every block of data. After writing to the appropriate registers to change the user bits for the next block, DITUSRBITAx and DITUSRBITBx must be written to enable the use of these bits. Note these registers are used in standalone mode only.

There are six user bits buffer registers associated with subframe A (left channel) and six user bits buffer registers associated with subframe B (right channel). These registers are listed with their locations in [Table A-107](#) and [Table A-108](#).

Table A-107. DITUSRBITAx Registers (RW)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITUSRBITA0	BYTE0	BYTE1	BYTE2	BYTE3
DITUSRBITA1	BYTE4	BYTE5	BYTE6	BYTE7
DITUSRBITA2	BYTE8	BYTE9	BYTE10	BYTE11
DITUSRBITA3	BYTE12	BYTE13	BYTE14	BYTE15
DITUSRBITA4	BYTE16	BYTE17	BYTE18	BYTE19
DITUSRBITA5	BYTE20	BYTE21	BYTE22	BYTE23

Peripherals Routed Through the DAI

Table A-108. DITUSRBITBx Registers (RW)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DITUSRBITB0	BYTE0	BYTE1	BYTE2	BYTE3
DITUSRBITB1	BYTE4	BYTE5	BYTE6	BYTE7
DITUSRBITB2	BYTE8	BYTE9	BYTE10	BYTE11
DITUSRBITB3	BYTE12	BYTE13	BYTE14	BYTE15
DITUSRBITB4	BYTE16	BYTE17	BYTE18	BYTE19
DITUSRBITB5	BYTE20	BYTE21	BYTE22	BYTE23

User Bit Update Register (DITUSRUPD)

This register is a 1-bit wide register (WO). After writing to the user bits registers (DITURSBITAX and DITUSRBITBx), a value of 0x1 must be written into DITUSRUPD register to enable the use of these bits in the next block of transfer.

Receiver Registers

The following sections describe the receiver registers.

Receive Control Register (DIRCTL)

This 32-bit register, described in [Table A-109](#) is used to set up error control and single-channel double-frequency mode.

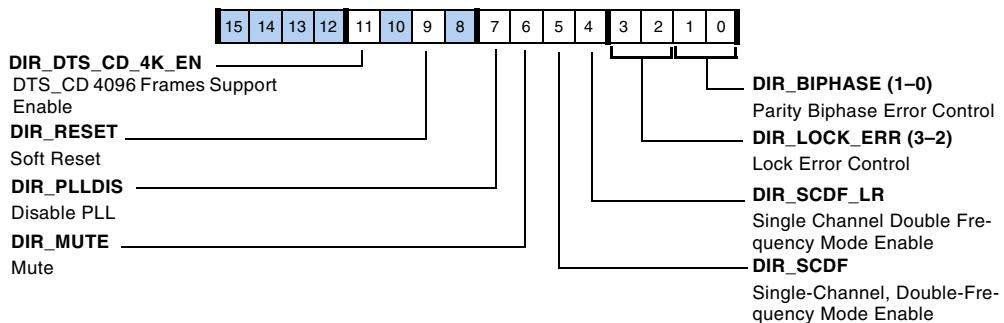


Figure A-108. DIRCTL Register

Table A-109. DIRCTL Register Bit Descriptions (RW)

Bit	Name	Description
1–0	DIR_BIPHASE	Parity Biphasic Error Control. When a parity or biphasic error occurs, the audio data will be handled according to these bits. 00 = No action taken 01 = Hold last valid sample 10 = Replace invalid sample with zeros 11 = Reserved
3–2	DIR_LOCK_ERR	Lock Error Control. When the DIR_LOCK bit in the DIRSTAT register is deasserted, it means the PLL has become unlocked and the audio data is handled according to these bit settings. 00 = No action taken 01 = Hold last valid sample 10 = Send zeros after the last valid sample 11 = Soft mute of the last valid audio is performed (as if NOSTREAM is asserted). This is valid only when linear PCM audio data is in the stream. When non-linear audio data is in the stream, this mode defaults to the case of DIR_LOCK_ERR1–0 bits = 10.
4	DIR_SCDF_LR	Single-Channel, Double-Frequency Channel Select. 0 = Left channel 1 = Right channel
5	DIR_SCDF	Single-Channel, Double-Frequency Mode Enable. 0 = 2 channel mode enabled 1 = SCDF mode

Peripherals Routed Through the DAI

Table A-109. DIRCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
6	DIR_MUTE	Mute. 0 = Mute disabled 1 = Mute serial data outputs, maintaining clocks (digital black)
7	DIR_PLLDIS	Disable PLL. Determines clock input for S/PDIF receiver. 0 = Use derived clock from the digital PLL 1 = Use clock input from external PLL
8	Reserved	
9	DIR_RESET	Reset S/PDIF Receiver. By default, the S/PDIF receiver is always enabled. If this bit is set, the S/PDIF receiver and digital PLL are disabled.
10	Reserved	
11	DIR_DTS_CD_4K_EN	DTS_CD 4096 Frames Support Enable. If this bit is set, and if NON-AUDIO preamble is detected, then the DIR_NOAUDIOCLR bit is asserted high and remains high if another NON AUDIO preamble is detected within 4096 frames, otherwise it is cleared. The assertion and deassertion of DIR_NOAUDIO bit can generate the DIR_NOAUDIO_INT DAI interrupt, if unmasked in the DAI_IRPTL_FE/DAI_IRPTL_RE interrupt mask registers. This bit is supported with on-chip Digital PLL only. This bit is applicable only for the ADSP-2147x and ADSP-2148x processors.
31–12	Reserved	

Receive Status Register (DIRSTAT)

The Status register consists of status bits (VALIDITY, NONAUDIO, NOSTREAM, BIPHERR, PARITYERR and LOCK), indicate the status of various functions supported by S/PDIF Receiver. It also has the lower byte of the 40-bit channel status information. The VALIDITY, NOSTREAM, BIPHERR, PARITYERR and LOCK bits are sticky and cleared on read. This register also contains the lower byte of the 40-bit channel status information. The bit settings for these registers are shown in [Figure A-109](#) and described in [Table A-110](#).

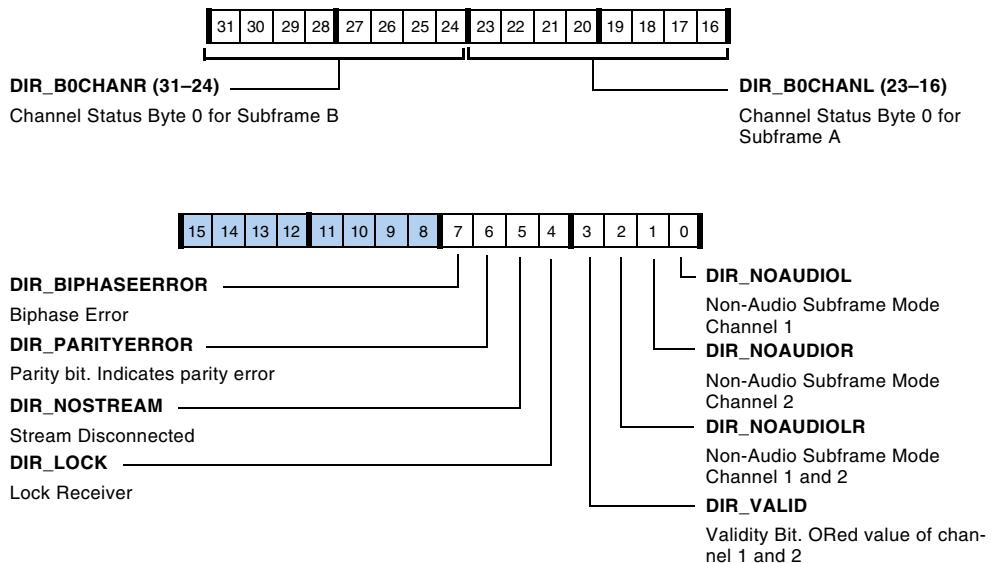


Figure A-109. DIRSTAT Register

Table A-110. DIRSTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	DIR_NOAUDIO	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_NOAUDIO	Non-Audio Subframe Mode Channel 2. Based on SMPTE 337M. 0 = Not non-audio subframe mode 1 = Non-audio subframe mode, channel 2
2	DIR_NOAUDIOLR	Non-Audio Frame Mode Channel 1 and 2. Based on SMPTE 337M. 0 = Not non-audio frame mode 1 = Non-audio frame mode
3 (ROC)	DIR_VALID	Validity Bit. ORed bits of channel 1 and 2. 0 = Linear PCM data 1 = Non-linear audio data

Peripherals Routed Through the DAI

Table A-110. DIRSTAT Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
4 (ROC)	DIR_LOCK	Lock Receiver. When set (=1), the digital PLL of receiver is locked, the corresponding DIR_LOCK bit is set. This bit can be polled to detect the DIR_LOCK condition. After the receiver is locked, the other status bits in DIRSTAT and the channel status (DIRCHANL/R) registers can be read. Interrupts can be also used with some status bits. 0 = Receiver not locked 1 = Receiver locked
5 (ROC)	DIR_NOSTREAM	No Stream Error. Asserted when the AES3/SPDIF stream is disconnected. When this bit is asserted and the audio data in the stream is linear PCM, the receiver performs a soft mute of the last valid sample from the AES3/SPDIF stream. A soft mute consists of taking the last valid audio sample and slowly and linearly decrementing it to zero, over a period of 4096 frames. During this time, the PLL three-states the charge pump until the soft mute has been completed. If non-linear PCM audio data is in the AES3/SPDIF stream when the NOSTREAM bit is asserted, the receiver sends out zeros after the last valid sample. 0 = Stream not disconnected 1 = Stream disconnected (default)
6 (ROC)	DIR_PARITYERROR	Parity Bit. When cleared, (=0), indicates that the AES3/SPDIF stream was received with the correct parity, or even parity. When set (=1), indicates that an error has occurred, and the parity is odd. 0 = No parity error 1 = Parity error
7 (ROC)	DIR_BIPHASEERROR	Biphase Error. When set (=1), indicates that a bi-phase error has occurred and the data sampled from the biphase stream may not be correct. 0 = No biphase error 1 = Biphase error
15–8	Reserved	
23–16	DIR_B0CHANL	Channel Status Byte 0 for Subframe A.
31–24	DIR_B0CHANR	Channel Status Byte 0 for Subframe B.

Receive Status Registers for Subframe A (DIRCHAN_A)

The S/PDIF receiver stores a maximum of 5 bytes (40-bit) status information. Note that status byte 0 is available in the DIRCTL register. This 32-bit register is described in [Table A-111](#).

Table A-111. DIRCHANAx Registers (RO)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DIRSTAT			BYTE0	
DIRCHAN _A	BYTE1	BYTE2	BYTE3	BYTE4

Receive Status Registers for Subframe B (DIRCHAN_B)

The S/PDIF receiver stores a maximum of 5 bytes (40-bit) status information. Note that status byte 0 is available in the DIRCTL register. This 32-bit register is described in [Table A-112](#).

Table A-112. DIRCHANBx Registers (RO)

Register	Bits 7–0	Bits 15–8	Bits 23–16	Bits 31–24
DIRSTAT				BYTE0
DIRCHAN _B	BYTE1	BYTE2	BYTE3	BYTE4

Real-Time Clock Registers

The following sections describe the registers associated with the real-time clock (RTC).

Peripherals Routed Through the DAI

Control Register (RTC_CTL)

This register, shown in [Figure A-110](#) and described in [Table A-113](#) control interrupt generation for the RTC.

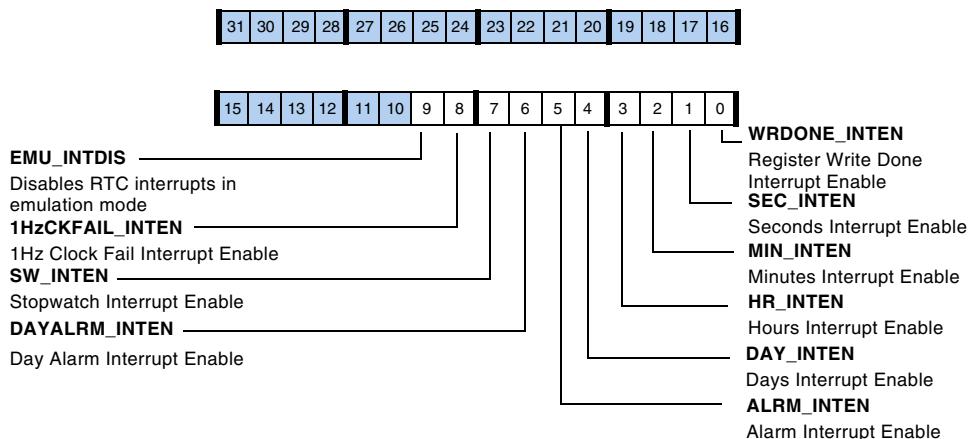


Figure A-110. RTC_CTL Register

Table A-113. RTC_CTL Register Bit Descriptions (RW)

Bit	Name	Description
0	WRDONE_INTEN	Register Write Done Interrupt Enable. 0 = Register Write Done interrupt disabled 1 = Register Write Done interrupt enabled
1	SEC_INTEN	Seconds Interrupt Enable. 0 = Seconds interrupt disabled 1 = Seconds interrupt enabled
2	MIN_INTEN	Minutes Interrupt Enable. 0 = Minutes interrupt disabled 1 = Minutes interrupt enabled
3	HR_INTEN	Hours Interrupt Enable. 0 = Hours interrupt disabled 1 = Hours interrupt enabled

Table A-113. RTC_CTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
4	DAY_INTEN	Days Interrupt Enable. 0 = Days interrupt disabled 1 = Days interrupt enabled
5	ALRM_INTEN	Alarm Interrupt Enable. 0 = Alarm interrupt disabled 1 = Alarm interrupt enabled
6	DAYALRM_INTEN	Day Alarm Interrupt Enable. 0 = Day alarm interrupt disabled 1 = Day alarm interrupt enabled
7	SW_INTEN	Stopwatch Interrupt Enable. 0 = Stopwatch interrupt disabled 1 = Stopwatch interrupt enabled
8	1HzCKFAIL_INTEN	RTC 1Hz Clock Fail Interrupt Enable. Indicates that the oscillator failed to start. 0 = RTC 1Hz clock fail interrupt disabled 1 = RTC 1Hz clock fail interrupt enabled
9	EMU_INTDIS	Disables/Enables RTC Interrupts in Emulation Mode. 0 = RTC interrupts enabled (if the individual interrupt enable bit is set) in emulation mode 1 = RTC interrupts disabled in emulation mode

Status Register (RTC_STAT)

This register, shown in [Figure A-111](#), The RTC Status register contains the RTC event flags and RTC interrupt status. These bits are sticky. Once set by the event, each bit remains set until cleared by a software read of this register. These sticky bits are independent of the interrupt enable bits in RTC_CTL register. Values are cleared by reading RTC_STAT register, except for the WR_PEND, ALRM_PEND and DAYALRM_PEND bits. Writes to any bit of this register has no effect. This register is cleared at reset.

Peripherals Routed Through the DAI

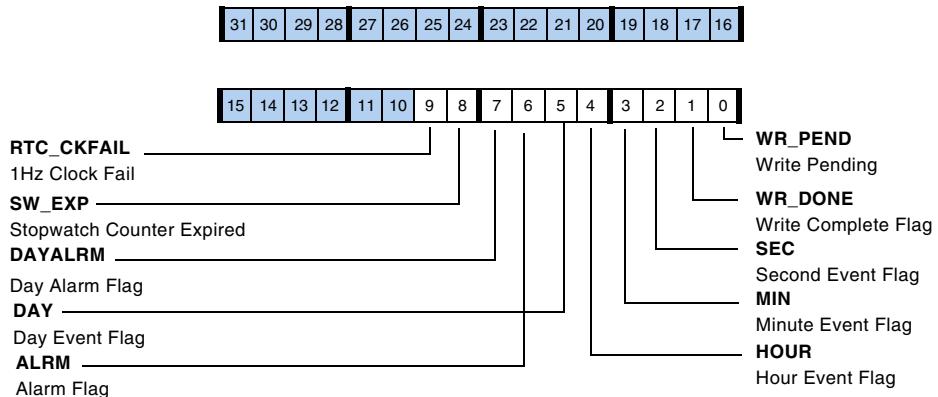


Figure A-111. RTC_STAT Register

Table A-114. RTC_STAT Register Bit Descriptions (RO)

Bit	Name	Description
0	WR_PEND	Register Write Pending. Shows that write to RTC_CLOCK, RTC_ALARM, RTC_SWTCH or RTC_INIT register is pending. This bit is automatically cleared/set by hardware.
1	SWR_DONE	RTC Register Write Done Flag. Shows that register write is over. Applicable only for alarm, clock and stopwatch registers. 1 = write to RTC_CLOCK, RTC_ALARM, RTC_SWTCH or RTC_INIT is over.
2	SEC	Second Event Flag. 0 = Second event has not occurred 1 = Second event has occurred
3	MIN	Minute Event Flag. 0 = Minute event has not occurred 1 = Minute event has occurred (clock counter value x:y:z:59)
4	HOUR	Hour Event Flag. 0 = Hour event has not occurred 1 = Hour event has occurred (clock counter value x:y:59:59)

Table A-114. RTC_STAT Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
5	DAY	Day Event Flag. 0 = Day event has not occurred 1 = Day event has occurred (clock counter value x:23:59:59)
6	ALRM	Alarm Flag. 0 = Daily alarm has not occurred 1 = Daily alarm has occurred
7	DAYALRM	Time of Day Flag. 0 = Alarm has not occurred 1 = Alarm has occurred
8	SW_EXP	Stop Watch Counter Expiration Flag. 0 = Stop watch counter has not expired 1 = Stop watch counter expired
9	RTC_CKFAIL	Disables/Enables RTC interrupts in emulation mode. 0 = RTC 1 Hz clock is functional 1 = RTC 1 Hz clock failed

Stopwatch Count Register (RTC_SWTCH)

This register, shown in [Figure A-112](#), contains the countdown value for the stop watch. The stopwatch counts down seconds from the programmed value and generates an interrupt (if `SW_INTEN` = 1) when the count reaches 0. The counter stops counting at this point and does not resume counting until a new nonzero value is written to this register. Writing a value of 0 to the running stopwatch forces it to stop; no interrupt is generated in this case. The register can be programmed to any value between 0 and $(2^{16} - 1)$ seconds (that is, a range of 18 hours, 12 minutes and 15 seconds).

Peripherals Routed Through the DAI

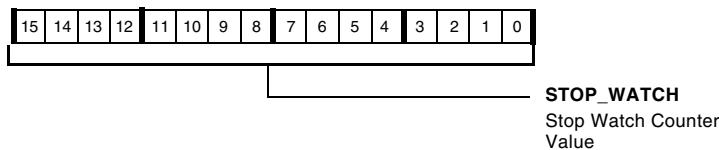


Figure A-112. RTC_SWTCH Register (RW)

Clock Register (RTC_CLOCK)

This register, shown in [Figure A-113](#), is used to read or write the current time. It has no reset and an undefined value when the module is first powered up. This register is updated every second. If RTC is already running when the core starts up, the values read from RTC_CLOCK are zero until the first second event occurs. In this case, programs must wait for the second event and then read the register. Writes of invalid time values are forbidden.

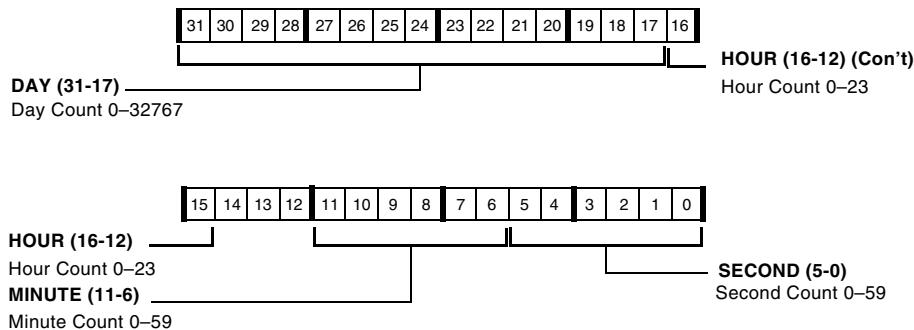


Figure A-113. RTC_CLOCK Register (RW)

Alarm Register (RTC_ALARM)

This register, shown in [Figure A-114](#), is programmed by software for the time (in hours, minutes, and seconds) the alarm interrupt occurs. Reads and writes can occur at any time. The alarm interrupt occurs whenever the hour, minute, and second fields first match those of the RTC status

register. The day interrupt occurs whenever the day, hour, minute, and second fields first match those of the RTC status register.

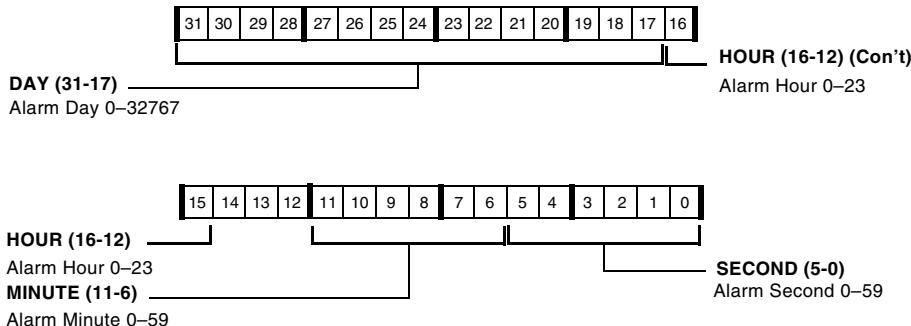


Figure A-114. RTC Alarm Register (RW)

Initialization Register (RTC_INIT)

This register, shown in [Figure A-115](#) and described in [Table A-115](#), provides the calibration function, powers down the unit, and grounds these buses.

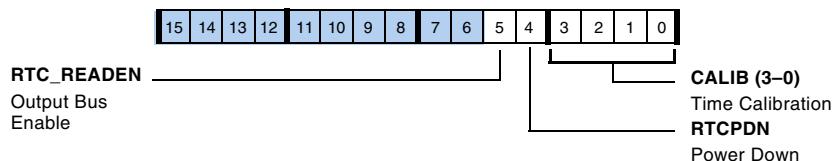


Figure A-115. RTC_INIT Register

Peripherals Routed Through the DAI

Table A-115. RTC_INIT Register Bit Descriptions (RW)

Bit	Name	Description
3–0	CALIB	Time Calibration. Max \pm 7 seconds. Calibration value is added/subtracted from the time every day at midnight. 1001 -> -1 second 1010 -> -2 second ----- 1111 -> -7 second 0001 -> +1 second 0010 -> +2 second ----- 0111 -> +7 second
4	RTCPDN	RTC Power Down. Active high. Write 1 to power down RTC oscillator, 0 to power up RTC oscillator. This bit must be compulsorily written once during first RTC oscillator power-up. 0 = RTC oscillator running 1 = Powers down RTC oscillator
5	RTC_READEN	Output Bus Enable. Enables the output buses between RTC I/O voltage portion and RTC core voltage portion. Setting this bit grounds these buses and prevents floating node consumption when the RTC is not used. At reset, the output buses are enabled (this bit is cleared). 0 = Buses are enabled (RTC in use) 1 = Buses are disabled (RTC not in use)

Initialization Status Register (RTC_INITSTAT)

Figure A-116 and Table A-116 describe the bits in the RTC_INITSTAT register.

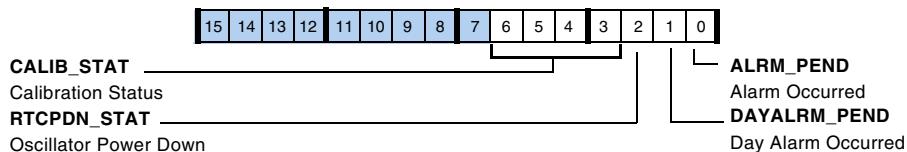


Figure A-116. RTC_INITSTAT Register

Table A-116. RTC_INITSTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	ALRM_PEND	Time Calibration. Indicates that an alarm has occurred. (Useful if core has powered down or reset in the middle). 0 = No daily alarm has occurred 1 = A daily alarm has occurred This bit is cleared on reading RTC_INITSTAT.
1	DAYALRM_PEND	RTC Power Down. Indicates that an alarm has occurred. (Useful if core has powered down or reset in the middle.) 0 = No day alarm has occurred 1 = A day alarm had occurred This bit is cleared on reading RTC_INITSTAT.
2	RTCPDN_STAT	Power Down Status. Status of RTC oscillator power-down bit. 0 = The RTC oscillator is running 1 = The RTC oscillator is powered down
6–3	CALIB_STAT	Calibration Status. Indicates whether CALIB value in the RTC_INIT register has been successfully programmed in the RTC. It should be equal to the value of CALIB.

Shift Register Register

The following sections provide bit information for the shift register register.

Control Register (SR_CTL)

This register, shown in [Figure A-117](#) and described in [Table A-117](#), enables the peripheral.

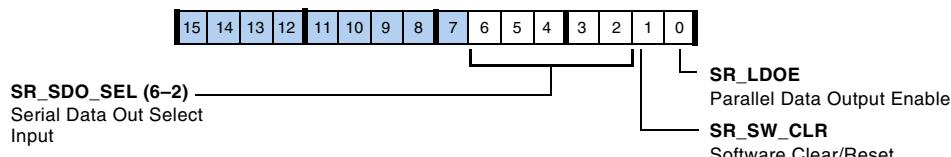


Figure A-117. SR_CTL Register

DPI Signal Routing Unit Registers

Table A-117. SR_CTL Register Bit Descriptions (RW)

Bit	Name	Description
0	SR_LDOE	Parallel Data Output Enable. This bit enables the parallel SR_LD017–0 output pins. It is cleared on chip reset (RESET) and/or asynchronously on external SR_CLR pin.
1	SR_SW_CLR	Software Clear/Reset. If this bit is 0, then the reset is active. 0 = Shift register cleared 1 = Shift register enabled
6–2	SR_SDO_SEL	Serial Data Out Multiplexer's Select Input. These bits select which parallel word is shifted through the SR_SDO pin. 00000 = LSB selected. 10001 = MSB selected.
31–7	Reserved	

DPI Signal Routing Unit Registers

The digital peripheral interface is comprised of a group of peripherals and the signal routing unit 2 (SRU2).

Miscellaneous Signal Routing Registers (SRU2_INPUTx, Group A)

Group A is used to route the 14 external pin signals to the inputs of the other peripherals. The MISCBx_0 outputs route to the interrupt latch bits or the pin buffer enable signals (PBEN).

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 A-118](#) through [Figure A-123](#) and [Table A-118](#).

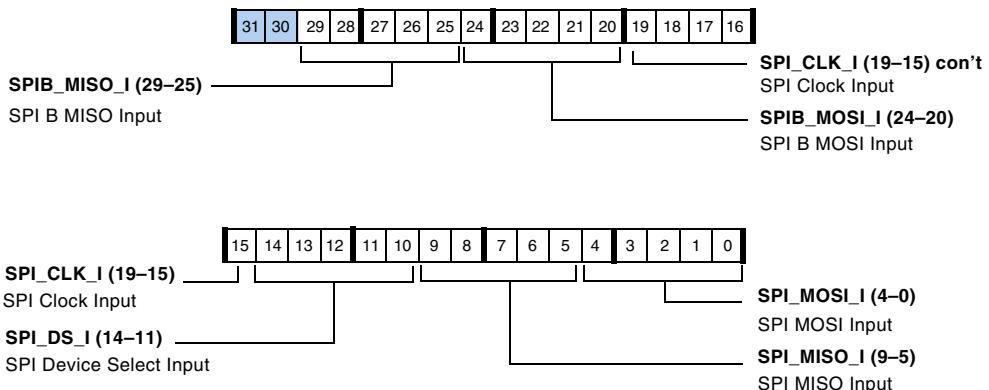


Figure A-118. SRU2_INPUT0 Register (RW)

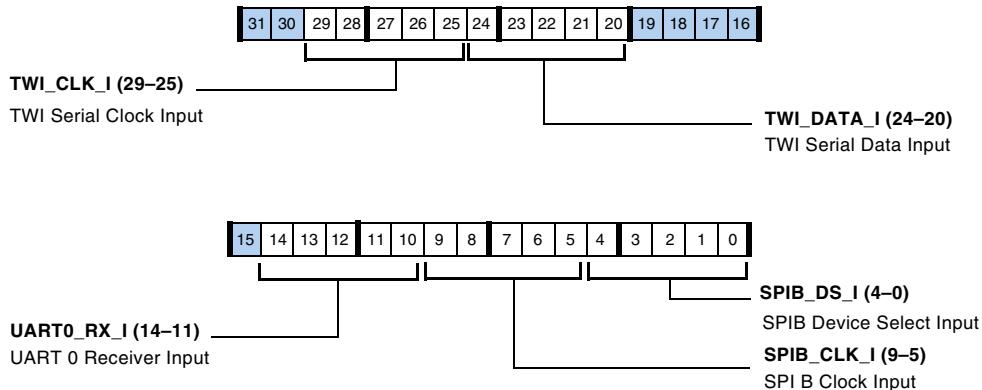


Figure A-119. SRU2_INPUT1 Register (RW)

DPI Signal Routing Unit Registers

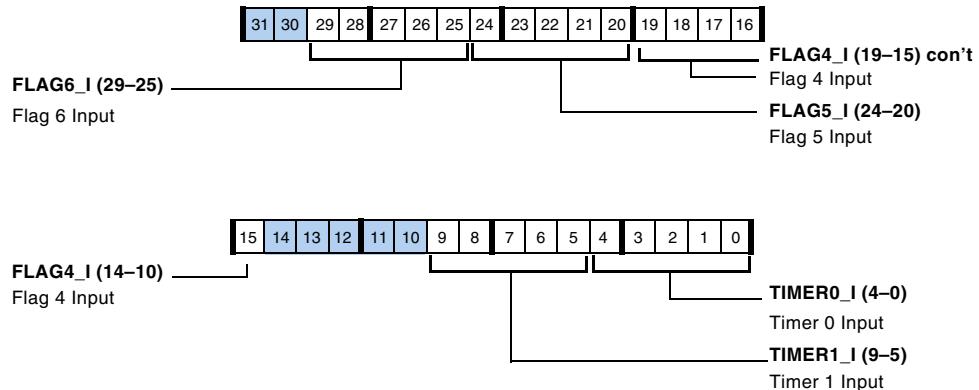


Figure A-120. SRU2_INPUT2 Register (RW)

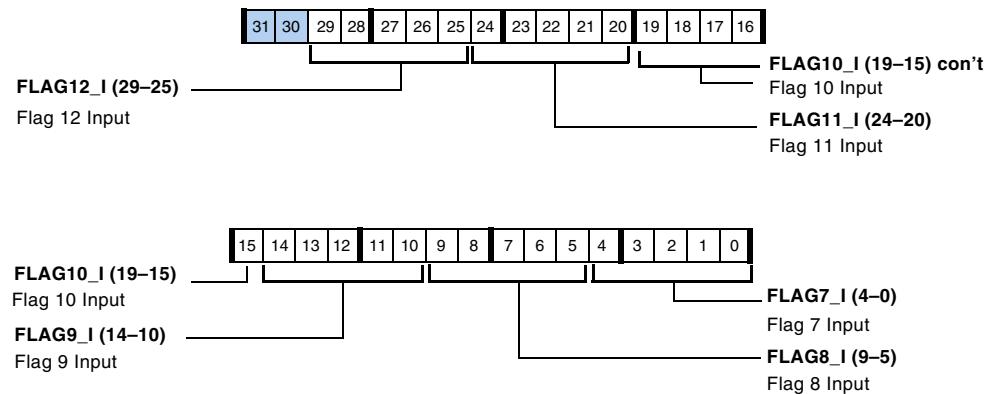


Figure A-121. SRU2_INPUT3 Register (RW)

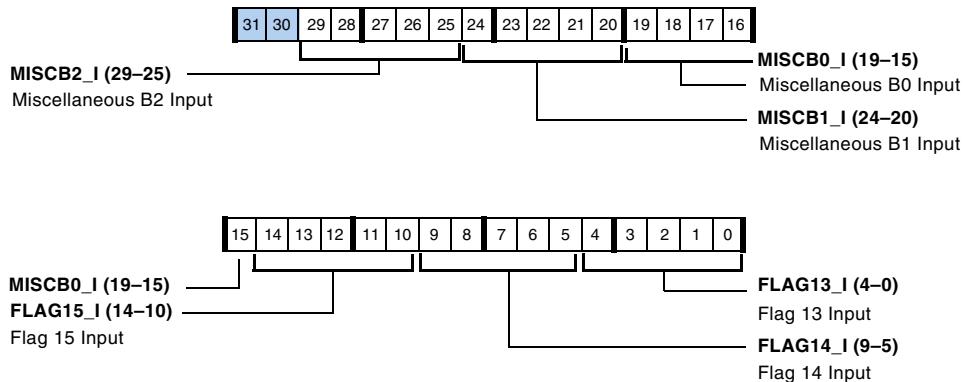


Figure A-122. SRU2_INPUT4 Register (RW)

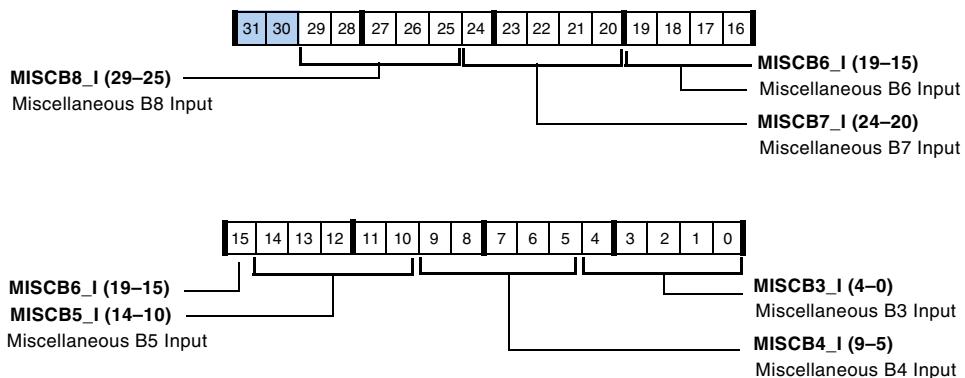


Figure A-123. SRU2_INPUT5 Register (RW)

Table A-118. Group A Connections

Selection Code	Signal	Description (Source Selection)
00000 (0x0)	LOW	Logic Level Low (0)
00001 (0x1)	HIGH	Logic Level High (1)
00010 (0x2)	DPI_PB01_O	External Pin 1
00011 (0x3)	DPI_PB02_O	External Pin 2

DPI Signal Routing Unit Registers

Table A-118. Group A Connections (Cont'd)

Selection Code	Signal	Description (Source Selection)
00100 (0x4)	DPI_PB03_O	External Pin 3
00101 (0x5)	DPI_PB04_O	External Pin 4
00110 (0x6)	DPI_PB05_O	External Pin 5
00111 (0x7)	DPI_PB06_O	External Pin 6
01000 (0x8)	DPI_PB07_O	External Pin 7
01001 (0x9)	DPI_PB08_O	External Pin 8
01010 (0xA)	DPI_PB09_O	External Pin 9
01011 (0xB)	DPI_PB10_O	External Pin 10
01100 (0xC)	DPI_PB11_O	External Pin 11
01101 (0xD)	DPI_PB12_O	External Pin 12
01110 (0xE)	DPI_PB13_O	External Pin 13
01111 (0xF)	DPI_PB14_O	External Pin 14
10000 (0x10)	TIMER0_O	Timer0 Output
10001 (0x11)	TIMER1_O	Timer1 Output
10010 (0x12)	Reserved	
10011 (0x13)	UART0_TX_O	UART0 Transmitter Output
10100 (0x14)	Reserved	
10101-11111	Reserved	

Pin Assignment Signal Routing (SRU2_PINx, Group B)

Group B connections, shown in [Figure A-124](#) through [Figure A-126](#) and [Table A-119](#), are used to route output signals to the 14 DPI pins.



For the ADSP-2147x and ADSP-2148x processors, the outputs of PWM units 3–1 can be routed to the DPI pins. See locations 0x23 – 0x2E in [Table A-119](#).

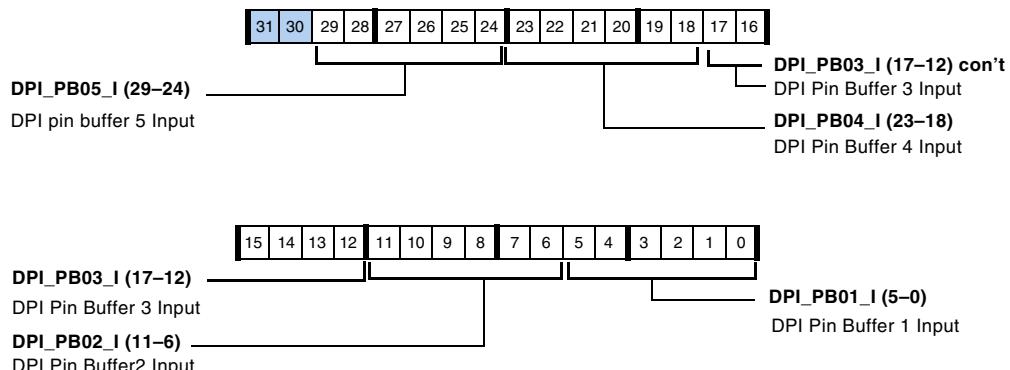


Figure A-124. SRU2_PIN0 Register

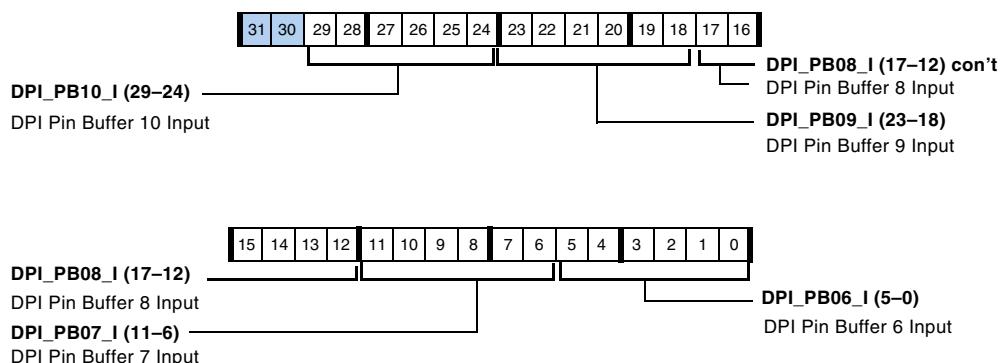


Figure A-125. SRU2_PIN1 Register

DPI Signal Routing Unit Registers

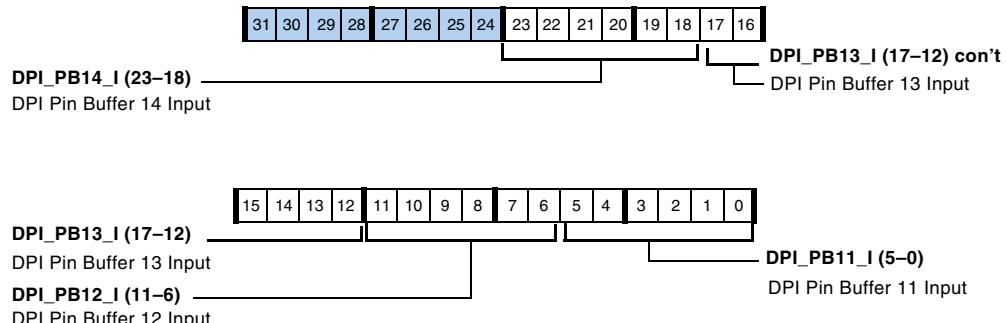


Figure A-126. SRU2_PIN2 Register

Table A-119. Group B Signals

Binary	Signal	Description (Source Selection)
000000 (0x0)	LOW	Logic Level Low (0)
000001 (0x1)	HIGH	Logic Level High (1)
000010 (0x2)	DPI_PB01_O	External Pin 1
000011 (0x3)	DPI_PB02_O	External Pin 2
000100 (0x4)	DPI_PB03_O	External Pin 3
000101 (0x5)	DPI_PB04_O	External Pin 4
000110 (0x6)	DPI_PB05_O	External Pin 5
000111 (0x7)	DPI_PB06_O	External Pin 6
001000 (0x8)	DPI_PB07_O	External Pin 7
001001 (0x9)	DPI_PB08_O	External Pin 8
001010 (0xA)	DPI_PB09_O	External Pin 9
001011 (0xB)	DPI_PB10_O	External Pin 10
001100 (0xC)	DPI_PB11_O	External Pin 11
001101 (0xD)	DPI_PB12_O	External Pin 12
001110 (0xE)	DPI_PB13_O	External Pin 13
001111 (0xF)	DPI_PB14_O	External Pin 14

Table A-119. Group B Signals (Cont'd)

Binary	Signal	Description (Source Selection)
010000 (0x10)	TIMER0_O	Timer0 Output
010001 (0x11)	TIMER1_O	Timer1 Output
010010 (0x12)	Reserved	
010011 (0x13)	UART0_TX_O	UART0 Transmitter Output
010100 (0x14)	Reserved	
010101 (0x15)	SPI_MISO_O	MISO from SPI
010110 (0x16)	SPI莫斯I_O	MOSI from SPI
010111 (0x17)	SPI_CLK_O	Clock Output from SPI
011000 (0x18)	SPI_FLG0_O	Slave Select 0 from SPI
011001 (0x19)	SPI_FLG1_O	Slave Select 1 from SPI
011010 (0x1A)	SPI_FLG2_O	Slave Select 2 from SPI
011011 (0x1B)	SPI_FLG3_O	Slave Select 3 from SPI
011100 (0x1C)	SPIB_MISO_O	MISO from SPIB
011101 (0x1D)	SPIB_MOSI_O	MOSI from SPIB
011110 (0x1E)	SPIB_CLK_O	Clock Output from SPIB
011111 (0x1F)	SPIB_FLG0_O	Slave Select 0 from SPIB
100000 (0x20)	SPIB_FLG1_O	Slave Select 1 from SPIB
100001 (0x21)	SPIB_FLG2_O	Slave Select 2 from SPIB
100010 (0x22)	SPIB_FLG3_O	Slave Select 3 from SPIB
100011 (0x23)	FLAG4_O	Flag/PWM 4 Output ¹
100100 (0x24)	FLAG5_O	Flag/PWM 5 Output
100101 (0x25)	FLAG6_O	Flag/PWM 6 Output
100110 (0x26)	FLAG7_O	Flag/PWM 7 Output
100111 (0x27)	FLAG8_O	Flag/PWM 8 Output
101000 (0x28)	FLAG9_O	Flag/PWM 9 Output
101001 (0x29)	FLAG10_O	Flag/PWM 10 Output

DPI Signal Routing Unit Registers

Table A-119. Group B Signals (Cont'd)

Binary	Signal	Description (Source Selection)
101010 (0x2A)	FLAG11_O	Flag/PWM 11 Output
101011 (0x2B)	FLAG12_O	Flag/PWM 12 Output
101100 (0x2C)	FLAG13_O	Flag/PWM 13 Output
101101 (0x2D)	FLAG14_O	Flag/PWM 14 Output
101110 (0x2E)	FLAG15_O	Flag/PWM 15 Output
101111 (0x2F)	PCG_CLKC_O	Precision Clock Generator Clock C Out
110000 (0x30)	PCG_CLKD_O	Precision Clock Generator Clock D Out
110001 (0x31)	PCG_FSC_O	Precision Clock Generator Frame Sync C Out
110010 (0x32)	PCG_FSD_O	Precision Clock Generator Frame Sync D Out
110011–111111	Reserved	

1 PWM not available on ADSP-2146x models.

Pin Enable Signal Routing (SRU2_PBENx, Group C)

Group C signals, shown in [Table A-120](#), are used to specify whether each DPI pin is used as an output or an input by setting the source for the pin buffer enable. When a pin buffer enable (`DPI_PBENxx_I`) is set (= 1), the signal present at the corresponding pin buffer input (`DPI_PBxx_I`) is driven off chip as an output. When a pin buffer enable is cleared (= 0), the signal present at the corresponding pin buffer input is ignored.

The pin enable control registers activate the drive buffer for each of the 14 DPI pins. When the pins are not enabled (driven), they can be used as inputs.

The registers that control group C settings are shown in [Figure A-127](#) through [Figure A-129](#).



The TWI output must operate as an open-drain output, the DPI input pins used for TWI data and clock should be connected to logic level 0.

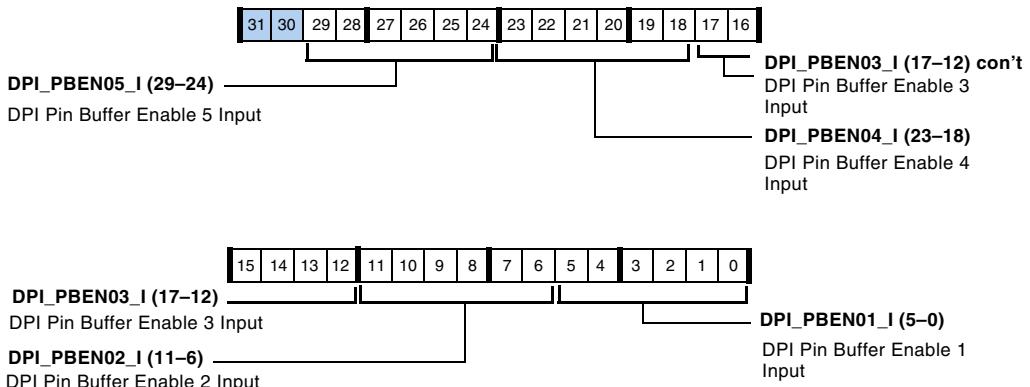


Figure A-127. SRU2_PBEN0 Register

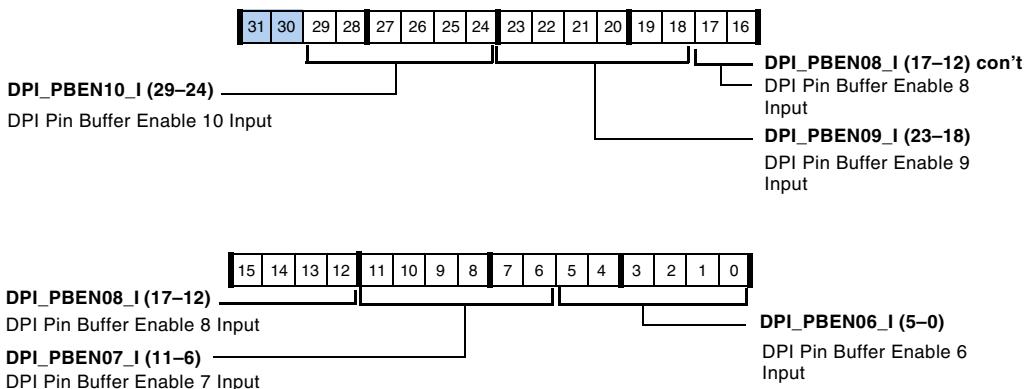


Figure A-128. SRU2_PBEN1 Register

DPI Signal Routing Unit Registers

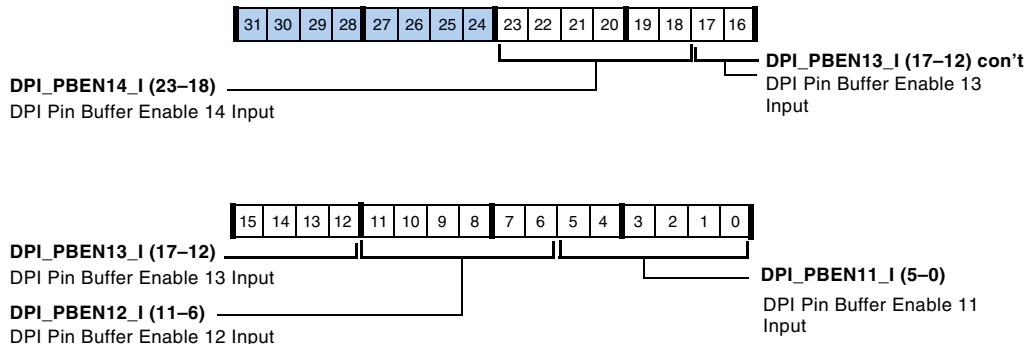


Figure A-129. SRU2_PBEN2 Register

Table A-120. Group C Signals

Binary	Signal	Description (Source Selection)
000000 (0x0)	LOW	Logic Level Low (0)
000001 (0x1)	HIGH	Logic Level High (1)
000010 (0x2)	MISCB0_O	Miscellaneous Control 0
000011 (0x3)	MISCB1_O	Miscellaneous Control 1
000100 (0x4)	MISCB2_O	Miscellaneous Control 2
000101 (0x5)	TIMER0_PBEN_O	Enable for Timer 0 Output
000110 (0x6)	TIMER1_PBEN_O	Enable for Timer 1 Output
000111 (0x7)	Reserved	
001000 (0x8)	UART0_TX_PBEN_0	Pin Enable for UART 0 Transmitter
001001 (0x9)	Reserved	
001010 (0xA)	SPIMISO_PBEN_O	Pin Enable for MISO from SPI
001011 (0xB)	SPIOMOSI_PBEN_O	Pin Enable for MOSI from SPI
001100 (0xC)	SPICLK_PBEN_O	Pin Enable for CLK from SPI
001101 (0xD)	SPIFLG0_PBEN_O	Pin Enable for Slave Select 0 from SPI
001110 (0xE)	SPIFLG1_PBEN_O	Pin Enable for Slave Select 1 from SPI
001111 (0xF)	SPIFLG2_PBEN_O	Pin Enable for Slave Select 2 from SPI

Table A-120. Group C Signals (Cont'd)

Binary	Signal	Description (Source Selection)
010000 (0x10)	SPIFLG3_PBEN_O	Pin Enable for Slave Select 3 from SPI
010001 (0x11)	SPIBMISO_PBEN_O	Pin Enable for MISO from SPIB
010010 (0x12)	SPIBMOSI_PBEN_O	Pin Enable for MOSI from SPIB
010011 (0x13)	SPIBCLK_PBEN_O	Pin Enable for CLK from SPIB
010100 (0x14)	SPIBFLG0_PBEN_O	Pin Enable for Slave Select 0 from SPIB
010101 (0x15)	SPIBFLG1_PBEN_O	Pin Enable for Slave Select 1 from SPIB
010110 (0x16)	SPIBFLG2_PBEN_O	Pin Enable for Slave Select 2 from SPIB
010111 (0x17)	SPIBFLG3_PBEN_O	Pin Enable for Slave Select 3 from SPIB
011000 (0x18)	FLAG4_PBEN_O	Pin Enable for Flag 4 Output
011001 (0x19)	FLAG5_PBEN_O	Pin Enable for Flag 5 Output
011010 (0x1A)	FLAG6_PBEN_O	Pin Enable for Flag 6 Output
011011 (0x1B)	FLAG7_PBEN_O	Pin Enable for Flag 7 Output
011100 (0x1C)	FLAG8_PBEN_O	Pin Enable for Flag 8 Output
011101 (0x1D)	FLAG9_PBEN_O	Pin Enable for Flag 9 Output
011110 (0x1E)	FLAG10_PBEN_O	Pin Enable for Flag 10 Output
011111 (0x1F)	FLAG11_PBEN_O	Pin Enable for Flag 11 Output
100000 (0x20)	FLAG12_PBEN_O	Pin Enable for Flag 12 Output
100001 (0x21)	FLAG13_PBEN_O	Pin Enable for Flag 13 Output
100010 (0x22)	FLAG14_PBEN_O	Pin Enable for Flag 14 Output
100011 (0x23)	FLAG15_PBEN_O	Pin Enable for Flag 15 Output
100100 (0x24)	TWI_DATA_PBEN_O	Data Output Enable from TWI
100101 (0x25)	TWI_CLK_PBEN_O	Clock Output Enable from TWI
100110 (0x26)	MISCB3_O	Miscellaneous Control 3
100111 (0x27)	MISCB4_O	Miscellaneous Control 4
101000 (0x28)	MISCB5_O	Miscellaneous Control 5
101001 (0x29)	MISCB6_O	Miscellaneous Control 6

DPI Signal Routing Unit Registers

Table A-120. Group C Signals (Cont'd)

Binary	Signal	Description (Source Selection)
101010 (0x2A)	MISCB7_O	Miscellaneous Control 7
101011 (0x2B)	MISCB8_O	Miscellaneous Control 8
101100–111111	Reserved	

DPI Pin Buffer Registers

The `DPI_PIN_PULLUP` and `DPI_PIN_STAT` control and return status of the DAI pin buffers.

Pin Buffer Status Register (DPI_PIN_STAT)

This 16-bit, read-only register is shown in [Figure A-130](#). Bits 13–0 of this register indicate the status of `DPI_PB14–1`. Reads from bits 15–14 always return 0. This register is updated at up to the `PCLK/2` rate.

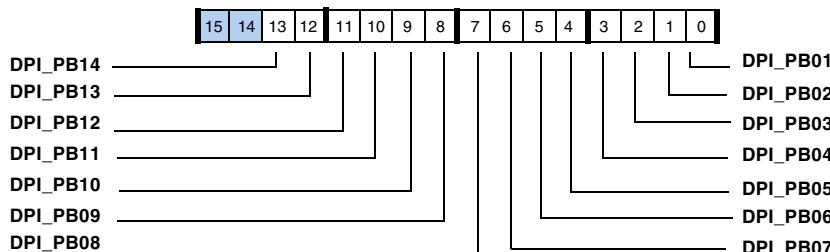


Figure A-130. `DPI_PIN_STAT` Register

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_IMASK` register contains the status on individual interrupts. Apart from the `DPI_IRPTL` register, there are two

additional registers, `DPI_IMASK_RE` and `DPI_IMASK_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-131](#) and listed in [Table A-121](#). Note that for each of these registers the bit names and numbers are the same.

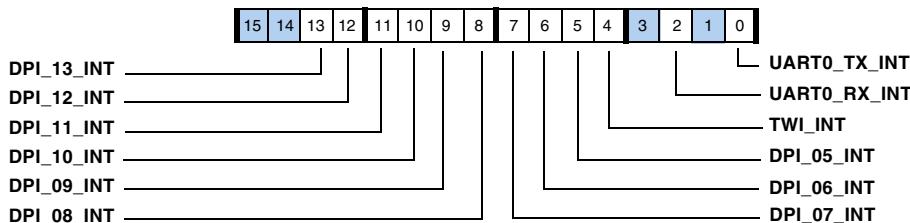


Figure A-131. DPI Interrupt Latch/Mask Register

Table A-121. DPI Interrupt Registers

Register	Description
<code>DPI_IRPTL</code> (ROC)	Interrupt Latch Register
<code>DPI_IRPTL_SH</code> (RO)	DPI_IMASK Shadow Register. Reads of this register returns data in DPI_IMASK without clearing contents of the register.
<code>DPI_IMASK_RE</code> (RW)	Rising Edge Interrupt Mask Register
<code>DPI_IMASK_FE</code> (RW)	Falling Edge Interrupt Mask Register

Peripherals Routed Through the DPI

The following sections provide information on the peripherals that are explicitly routed through the digital peripheral interface.

Serial Peripheral Interface Registers

The following sections describe the registers associated with the two serial peripheral interfaces (SPIs). Note that the SPI port is routed through the DPI.

Control Registers (SPICTL, SPICTLB)

The SPI control (SPICTL) registers are used to configure and enable the SPI system. The bit settings for these registers are shown in [Figure A-132](#) and described in [Table A-122](#).

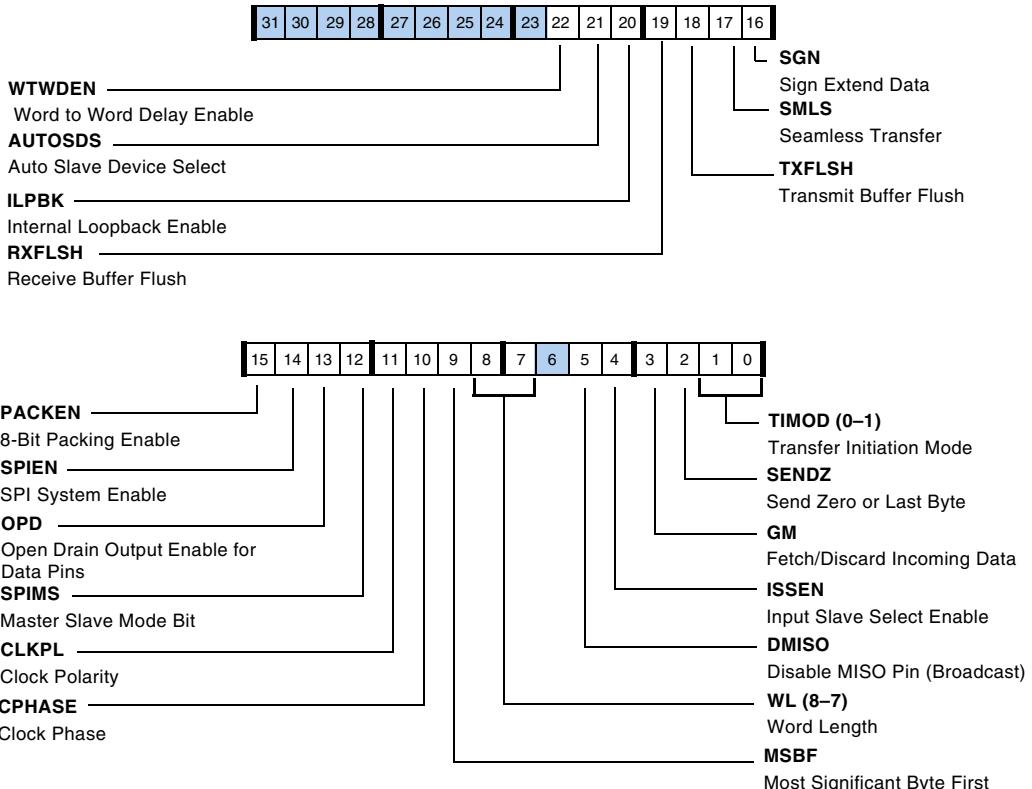


Figure A-132. SPICTL, SPICTLB Registers

Table A-122. SPICTL Register Bit Descriptions (RW)

Bit	Name	Description
1–0	TIMOD	<p>Transfer Initiation Mode. Defines transfer initiation mode and interrupt generation.</p> <p>00 = Initiate transfer by read of receive buffer. Interrupt active when receive buffer is full.</p> <p>01 = Initiate transfer by write to transmit buffer. Interrupt active when transmit buffer is empty.</p> <p>10 = Enable DMA transfer mode. Interrupt configured by DMA.</p> <p>11 = Reserved</p>
2	SENDZ	<p>Send Zero. Send zero or the last word when TXSPI is empty.</p> <p>0 = Send last word</p> <p>1 = Send zeros</p>
3	GM	<p>Get Data. When RXSPI is full, get data or discard incoming data.</p> <p>0 = Discard incoming data</p> <p>1 = Get more data, overwrites the previous data</p>
4	ISSEN	<p>Input Slave-Select Enable. Enables slave-select input ($\overline{\text{SPIDS}}$ pin) for the master. $\overline{\text{SPIDS}}$ operation depends on the SPI configuration. If the SPI is a slave, $\overline{\text{SPIDS}}$ acts as the slave-select input. The state of this input pin is observable in bit 7 of the SPIFLGx register.</p> <p>As master, $\overline{\text{SPIDS}}$ can serve as an error-detection input in a multimaster environment. The ISSEN-bit enables this feature. When ISSEN =1, the $\overline{\text{SPIDS}}$ input is the master mode error input; otherwise, $\overline{\text{SPIDS}}$ is ignored.</p> <p>0 = Disable</p> <p>1 = Enable</p>
5	DMISO	<p>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.</p> <p>Different CPUs or processors can take turns being master, and one master may simultaneously shift data into multiple slaves (broadcast mode).</p> <p>However, only one slave may drive its output to write data back to the master at any given time. This must be enforced in the broadcast mode, where several slaves can be selected to receive data from the master, but only one slave can be enabled to send data back to the master. The (DMISO) bit disables MISO as an output.</p> <p>0 = MISO enabled</p> <p>1 = MISO disabled</p>

Peripherals Routed Through the DPI

Table A-122. SPICTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
6	Reserved	
8–7	WL	<p>Word Length. SPI port can transmit and receive three word widths: 00 = 8 bits 01 = 16 bits 10 = 32 bits</p> <p>8-bit word. SPI port sends out only the lower eight bits of the word written to the SPI buffer. For example 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. This code works only if the MSBF bit is zero in both the transmitter and receiver, and the SPICLK frequency is small. If MSBF = 1 in the transmitter and receiver, and SPICLK has a small frequency, the received words follow the order 0x12, 0x34, 0x56, 0x78.</p> <p>16-bit word. When transmitting, the SPI port sends out only the lower 16 bits of the word written to the SPI buffer.</p> <p>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.</p> <p>32-bit word. No packing of the RXSPI or TXSPI registers is necessary as the entire 32-bit register is used for the data word.</p>
9	MSBF	<p>Most Significant Byte First.</p> <p>0 = LSB sent/received first 1 = MSB sent/received first</p>
10	CPHASE	<p>Clock Phase. Selects the transfer format.</p> <p>0 = SPICLK starts toggling at the middle of 1st data bit 1 = SPICLK starts toggling at the start of 1st data bit (default setting)</p>
11	CLKPL	<p>Clock Polarity.</p> <p>0 = Active high SPICLK (SPICLK low is the idle state) 1 = Active low SPICLK (SPICLK high is the idle state)</p> <p>Note that the CLKPL/CPHASE bits define the SPI mode.</p>
12	SPIMS	<p>SPI Master Select. Configures SPI module as master or slave.</p> <p>0 = Device is a slave device 1 = Device is a master device</p>

Table A-122. SPICTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
13	OPD	<p>Open Drain Output Enable. Enables open drain data output enable for MOSI and MISO pins.</p> <p>0 = Drive (MOSI and MISO driven) 1 = Open drain (MOSI and MISO three-stated).</p> <p>In a multimaster/slave SPI system, the data output pins (MOSI and MISO) can be configured to behave as open drain drivers to prevent contention and possible damage to pin drivers. An external pull-up resistor is required on both the MOSI/MISO pins when this option is selected.</p> <p>When the OPD bit is set and the SPI ports are configured as masters, the SPI_MOSI_O pin is three-stated. Instead the SPI_MOSI_PBEN_O pin is driven and act as output enable pin. Note that the corresponding DPI input buffer pin should be tied to GND.</p> <p>Similarly, when OPD is set and the SPI ports are configured as slaves, the SPI_MISO_O pin is three-stated. Instead the SPI_MISO_PBEN_O pin is driven and act as output enable pin. Note that the corresponding DPI input buffer pin should be tied to GND. See “Pin Buffers as Open Drain” on page 9-11.</p>
14	SPIEN	<p>SPI Port Enable. Enables the SPI port. If configured as a master (SPIMS=1) and SPIEN=0, the MOSI and SPICLK outputs are disabled, and the MISO input is ignored. If configured as a slave (SPIMS=0) and SPIEN=0, the MOSI and SPICLK inputs are ignored, and the MISO output is disabled. The SPIEN and SPIMS bits can be cleared by hardware if the MME-bit is set. For SPI slaves, the slave-select input ($\overline{\text{SPIIDS}}$) acts like a reset for the internal SPI logic. For this reason, the $\overline{\text{SPIIDS}}$ line must be error free.</p> <p>The SPIEN bit 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 SPISTATx registers. These bits remain set if they are already set.</p> <p>Note: always clear the W1C-type bits in SPISTATx registers before re-enabling the SPI, as these bits do not get cleared even if the SPI is disabled. This can be done by writing 0xFF to the SPISTATx registers. In the case of an MME error, enable the SPI ports after $\overline{\text{SPIIDS}}$ is deasserted.</p> <p>0 = SPI module is disabled 1 = SPI module is enabled</p>

Peripherals Routed Through the DPI

Table A-122. SPICTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
15	PACKEN	<p>Packing Enable. The SPI unpacks data when it transmits and packs data when it receives. In order to communicate with 8-bit SPI devices and store 8-bit words in internal memory, a packed transfer feature is built into the SPI port.</p> <p>0 = No packing 1 = 8 to 16-bit packing</p> <p>Note: This bit may be 1 only when WL = 00 (8-bit transfer). When in transmit mode, the PACKEN bit unpacks data. When packing is enabled, two 8-bit words are packed into one 32-bit word. When the SPI port is transmitting, two 8-bit words are unpacked from one 32-bit word. When receiving, words are packed into one 32-bit word from two 8-bit words.</p> <p>The value 0XXLMXXJK (where XX is any random value and JK and LM are data words to be transmitted out of the SPI port) is written to the TXSPI register. The processor transmits 0xJK first and then transmits 0xLM.</p> <p>The receiver packs the two words received, 0xJK and then 0xLM, into a 32-bit word. They appear in the RXSPI register as:</p> <p>0x00LM00JK => if SGN is configured to 0 or L, J < 7 0xFFLMFFJK => if SGN is configured to 1 and L, J > 7</p>
16	SGN	<p>Sign Extend.</p> <p>0 = No sign extension 1 = Sign extension</p>
17	SMLS	<p>Seamless Transfer.</p> <p>0 = Seamless transfer disabled. After each word transfer there is a delay before the next word transfer starts. The delay is 2.5 SPICLK cycles</p> <p>1 = Seamless transfer enabled. There is no delay before the next word starts, a seamless operation. Not supported in mode TIMOD1-0 = 00 and CPHASE=0 for all modes.</p>
18	TXFLSH	<p>Flush Transmit Buffer. Write a 1 to clear TXSPI.</p> <p>0 = TXSPI not cleared 1 = TXSPI cleared</p>
19	RXFLSH	<p>Clear RXSPI. Write a 1 to clear RXSPI.</p> <p>0 = RXSPI not cleared 1 = RXSPI cleared</p>
20	ILPBK	<p>Internal Loop Back. This mode interconnects the MOSI with the MISO pins internally. In this mode the SPIMS bit must be set.</p> <p>0 = No internal loopback 1 = Internal loopback enabled</p>
31–23	Reserved	

DMA Configuration Registers (SPIDMAC, SPIDMACB)

These 17-bit SPI registers are used to control DMA transfers and are shown in [Figure A-133](#) and described in [Table A-123](#).

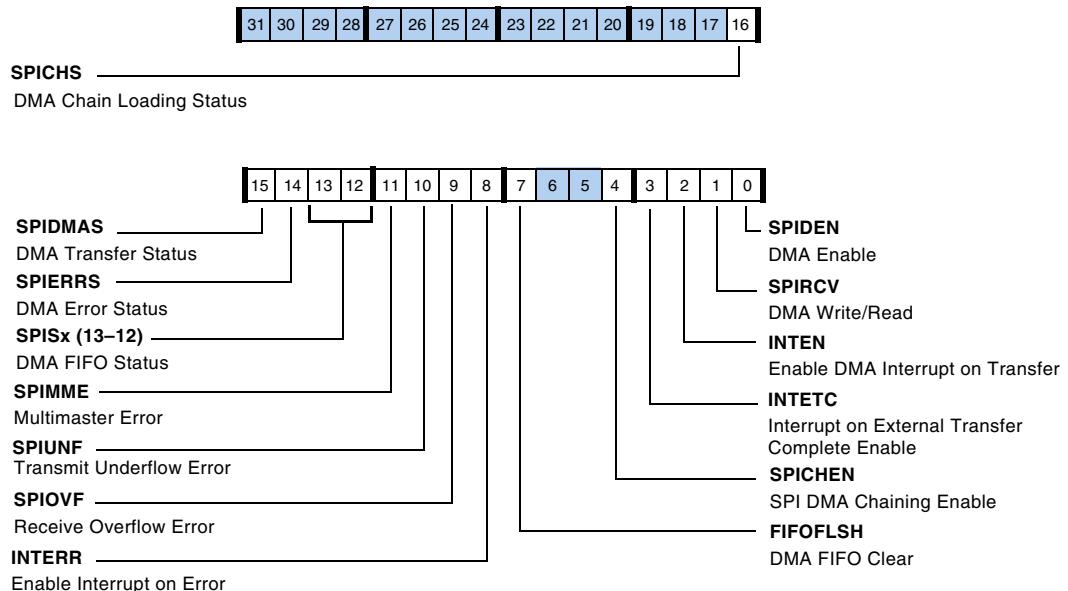


Figure A-133. SPIDMAC, SPIDMACB Registers

Table A-123. SPIDMAC, SPIDMACB Register Bit Descriptions (RW)

Bit	Name	Description
0	SPIDEN	DMA Enable. 0 = Disable 1 = Enable
1	SPIRCV	DMA Write/Read. 0 = SPI transmit (read from internal memory) 1 = SPI receive (write to internal memory)

Peripherals Routed Through the DPI

Table A-123. SPIDMAC, SPIDMACB Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
2	INTEN	Enable DMA Interrupt on Transfer. 0 = Disable 1 = Enable
3	INTETC	Interrupt on External Transfer Complete Enable. Selects interrupt event for transmit DMA 0 = DMA interrupt generated when DMA count reaches zero. 1 = DMA interrupt generated when last bit of last word is shifted out. Note: both INTEN and INTETC bits, when enabled, generate an interrupt for INTETC.
4	SPICHEN	SPI DMA Chaining Enable. 0 = Disable 1 = Enable
6–5	Reserved	
7 (WOC)	FIFOFLSH	DMA FIFO Clear. Clears the SPIS bit. 0 = Disable 1 = Enable
8	INTERR	Enable Interrupt on Error. 0 = Disable 1 = Enable
9 (RO)	SPIOVF	Receive OverFlow Error (SPIRCV = 1). 0 = Successful transfer 1 = Error – data received with RXSPI full
10 (RO)	SPIUNF	Transmit Underflow Error (SPIRCV = 0). 0 = Successful transfer 1 = Error occurred in transmission with no new data in TXSPI
11 (RO)	SPIMME	Multimaster Error. 0 = Successful transfer 1 = Error during transfer
13–12 (RO)	SPIS	DMA FIFO Status. 00 = FIFO empty 11 = FIFO full 10 = FIFO partially full 01 = Reserved

Table A-123. SPIDMAC, SPIDMACB Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
14 (RO)	SPIERRS	DMA Error Status. This bit is set if SPIOVF, SPIUNF or DPIMME bits are set. 0 = Successful DMA transfer 1 = Errors during DMA transfer
15 (RO)	SPIDMAS	DMA Transfer Status. 0 = DMA idle 1 = DMA in progress
16 (RO)	SPICHGS	DMA Chain Loading Status. 0 = Chain idle 1 = Chain loading in progress
31–17	Reserved	

Baud Rate Registers (SPIBAUD, SPIBAUDB)

These SPI registers are used to set the bit transfer rate for a master device. When configured as slaves, the value written to these registers is ignored. The (SPIBAUDx) registers can be read from or written to at any time. Bit descriptions are provided in [Table A-124](#). Note that the minimum value of BAUDR = 0x2, since the max SPICLK = PCLK/4 in master mode.

Table A-124. SPIBAUD, SPIBAUDB Register Bit Descriptions (RW)

Bit	Name	Description
0	Reserved	
15–1	BAUDR	Baud Rate Enable. Enables the SPICLK per the equation: SPICLK baud rate = PCLK / (4 x BAUDR) Default = 0
19–16	Reserved	
25–20	STDC	Sequential Transfer Delay. The word to word delay(T4) = 1.5 SPI CLK Period + T3 and T3 = 0.5 SPICLK period for STDC = 0. T3 = STDC × SPICLK period for STDC > 0.
31–26	Reserved	

Peripherals Routed Through the DPI

Status (SPISTAT, SPISTATB) Registers

The SPISTAT and SPISTATB registers are used to detect when an SPI transfer is complete, if transmission/reception errors occur, and the status of the TXSPI and RXSPI FIFOs. The bit settings for these registers are shown in [Figure A-134](#) and described in [Table A-125](#).

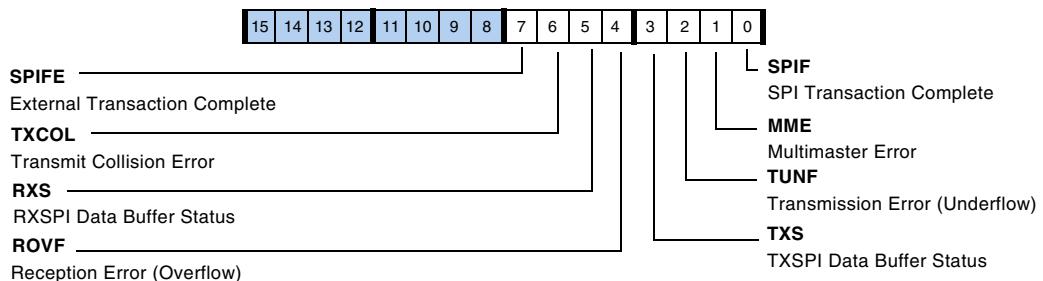


Figure A-134. SPISTAT, SPISTATB Registers

Table A-125. SPISTAT Register Bit Descriptions (RO)

Bit	Name	Description
0 (RO)	SPIF	SPI Transmit or Receive Transfer Complete. SPIF is set when an SPI single-word transfer is complete.
1 (W1C)	MME	Multimaster Error or Mode-Fault Error. MME is set in a master device when some other device tries to become the master. In multimaster mode, if the <u>SPIIDS</u> input signal of a master is asserted (low) an error has occurred. This means that another device is also trying to be the master. Clears the SPIMME bit.
2 (W1C)	TUNF	Transmission Error. TUNF is set when transmission occurred with no new data in TXSPI register. The TUNF bit (2) is set when all of the conditions of transmission are met and there is no new data in TXSPI (TXSPI is empty). In this case, the transmission contents depend on the state of the SENDZ bit in the SPICTL register. Clears the SPIUNF bit.

Table A-125. SPISTAT Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
3 (RO)	TXS	Transmit Data Buffer Status. TXSPI data buffer status. 0 = Empty 1 = Full
4 (W1C)	ROVF	Reception Error. ROVF is set when data is received with receive buffer full.
5 (RO)	RXS	Receive Data Buffer Status. The ROVF flag (bit 4) is when a new transfer has completed before the previous data is read from the RXSPI register. This bit indicates that a new word was received while the receive buffer was full. The ROVF flag is cleared by a 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. 0 = Empty 1 = Full
6 (W1C)	TXCOL	Transmit Collision Error. When TXCOL is set, it is possible that corrupt data was transmitted. The TXCOL flag (bit 6) is set when a write to the TXSPI register coincides with the load of the shift register. The write to TXSPI can be via the software or the DMA. This bit indicates that corrupt data may have been loaded into the shift register and transmitted. In this case, the data in TXSPI may not match what was transmitted. This error can easily be avoided by proper software control. The TXCOL bit is cleared by a W1C-type software operation. Note that this bit is never set when the SPI is configured as a slave with CPHASE = 0. The collision may occur, but it cannot be detected.
7 (RO)	SPIFE	External Transaction Complete. Set (= 1) when the SPI transaction is complete on the external interface. This bit is very useful in DMA mode showing that the peripheral has completed all the external transfers corresponding to the DMA programmed. For more information, see “ Transfer Initiate Mode ” on page 15-13 and “ DMA Transfers ” on page 15-21.
31–8	Reserved	

Peripherals Routed Through the DPI

SPI Port Flags Registers (SPIFLG, SPIFLGB)

The SPIFLG and SPIFLGB registers are used to enable individual SPI slave-select lines when the SPI is enabled as a master. This register is ignored if the SPI is programmed as a slave. The bit settings for these registers are shown in [Figure A-135](#) and described in [Table A-126](#).

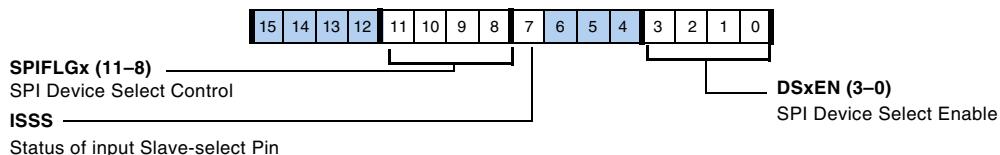


Figure A-135. SPIFLG, SPIFLGB Registers

Table A-126. SPIFLG, SPIFLGB Register Bit Descriptions (RW)

Bit	Name	Description
3–0	DSxEN	SPI Device Select Enable. Enables or disables the corresponding output signal to the SRU2 be used for SPI slave-select. 0 = Disable SPIFLGx output enable 1 = Enable SPIFLGx output enable Note DS0EN bit is set in SPI master mode only.
6–4	Reserved	
7 (RO)	ISSS	Input Slave Service Select. Reflects the service selection for the slave-select input pin ($\overline{\text{SPIDS}}$). 0 = $\overline{\text{SPIDS}}$ pin ignored 1 = $\overline{\text{SPIDS}}$ pin used as multimaster error detection (default for SPI only)
11–8	SPIFLGx	SPI Device Select Control. Selects (if cleared, = 0) a corresponding DPI pin (depending on pin routing) output to be asserted for an SPI slave-select. For AUTOSDS=1, there is automatic HW control regardless of CPHASE setting 0000 = All SPIFLGx cleared 1111 = All SPIFLGx set (default)
12–31	Reserved	

UART Control and Status Registers

The processor provides a set of PC-style, industry-standard control and status registers for each UART. These IOP registers are byte-wide registers that are mapped as half-words with the most significant byte zero-filled.

Line Control Register (UART0LCR)

The UART line control register (UART0LCR, shown in [Figure A-136](#) and described in [Table A-127](#)) controls the format of received and transmitted character frames.



Some UART registers share the same IOP address. The **UART0DLL** registers are mapped to the same address as the **UARTxTHR** and **UART0BR** registers. The **UART0DLH** registers are mapped to the same address as the interrupt enable registers (**UART0IER**). Note that the **UARTDLAB** bit in the **UART0LCR** register must be set before the UART divisor latch registers can be accessed. If the **UARTDLAB** bit is cleared, access to the **UART0THR** and **UART0BR** or **UART0IER** registers occurs.

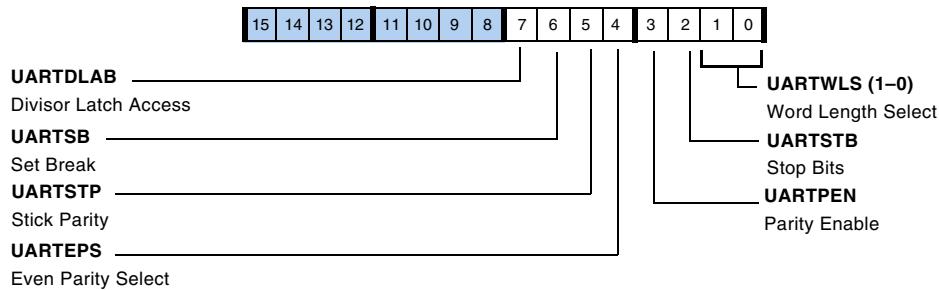


Figure A-136. UART0LCR Register

Peripherals Routed Through the DPI

Table A-127. UART0LCR Register Bit Descriptions (RW)

Bit	Name	Description
1–0	UARTWLS	Word Length Select. 00 = 5-bit word(UARTWLS5) 01 = 6-bit word(UARTWLS6) 10 = 7-bit word(UARTWLS7) 11 = 8-bit word(UARTWLS8)
2	UARTSTB	Stop Bits. 0 = 1 stop bit 1 = 2 stop bits for non-5-bit word length or 1 1/2 stop bits for 5-bit word length
3	UARTPEN	Parity Enable. 0 = Parity not transmitted or checked 1 = Transmit and check parity
4	UARTEPS	Even Parity Select. 0 = Odd parity when PEN = 1 and STP = 0 1 = Even parity when PEN = 1 and STP = 0
5	UARTSTP	Stick Parity. Forces parity to defined value if set and PEN = 1. 0 = Parity transmitted and checked as 1 1 = Parity transmitted and checked as 0
6	UARTSB	Set Break. The UART transmit pin is driven high normally. This bit is used to force the transmit pin to zero. This bit functions even when the UART is not enabled. Using this bit the UART TX pin can be used as a flag pin when the UART is not used. 0 = No force 1 = Force UART0Tx_O pin to 0.
7	UARTDLAB	Divisor Latch Access Bit. Because some IOP registers share the same address, this bit provides access as follows. 0 = Enable access to UART0THR, UART0RBR, and UART_IER registers 1 = Enable access to UART0DLL and UARTxDLH registers

Line Status Register (UART0LSR)

The UART line status register (UART0LSR) contains UART status information as shown in [Figure A-137](#) and described in [Table A-128](#). There is also a shadow register, UART0LSRSH, that allows programs to read the contents of the corresponding main register without affecting the status the UART.

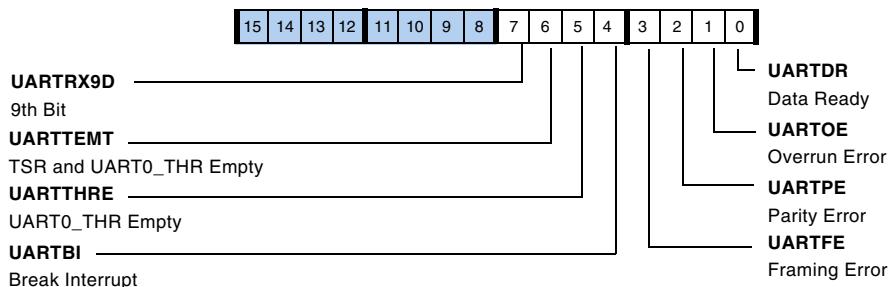


Figure A-137. UART0LSR Register

Table A-128. UART0LSR Register Bit Descriptions

Bit	Name	Description
0	UARTDR	Data Ready. This bit is cleared when the UART receive buffer (UARTxRBR) is read. 0 = No new data 1 = UARTx_RBR holds new data
1 (ROC ¹)	UARTOE	Overrun Error. 0 = No overrun 1 = UARTx_RBR overwritten before read
2 (ROC ¹)	UARPE	Parity Error. 0 = No parity error 1 = Parity error
3 (ROC ¹)	UARTFE	Framing Error. 0 = No error 1 = Invalid stop bit error

Peripherals Routed Through the DPI

Table A-128. UART0LSR Register Bit Descriptions (Cont'd)

Bit	Name	Description
4	UARTBI	Break Interrupt. The break interrupt (UARTBI), overrun error (UARTOE), parity error (UARTPE), and framing error (UARTFE) bits are cleared when the UART line status register (UART0LSR) is read. The data ready (UARTDNR) bit is cleared when the UART receive buffer register (UART0RBR) is read. 0 = No break interrupt 1 = Break interrupt. This indicates Rx pin was held low for more than the max word length.
5	UARTTHRE	UARTx_THR Empty. The UARTTHRE bit indicates that the UART transmit channel is ready for new data, and software can write to the UARTxTHR register. Writes to UART0THR clear the UARTTHRE bit. It is set again when data is copied from UART0THR to the transmit shift register (UART0TSR). The UARTTEMT bit can be evaluated to determine whether a recently initiated transmit operation has been completed. 0 = Not empty 1 = Empty (default)
6	UARTTEMT	TSR and UART0_THR Empty. 0 = Full 1 = Both empty
7	UARTRX9D	9th bit of the received character-address detect

1 These bits are read-only in the UARTxLRSH (shadow) register.

Interrupt Enable Register (UART0IER)

The interrupt enable register (shown in [Figure A-138](#)) is used to enable requests for system handling of empty or full states of UART data registers. Unless polling is used as a means of action, the UARTRBFIE and/or UARTTBEIE bits in this register are normally set.

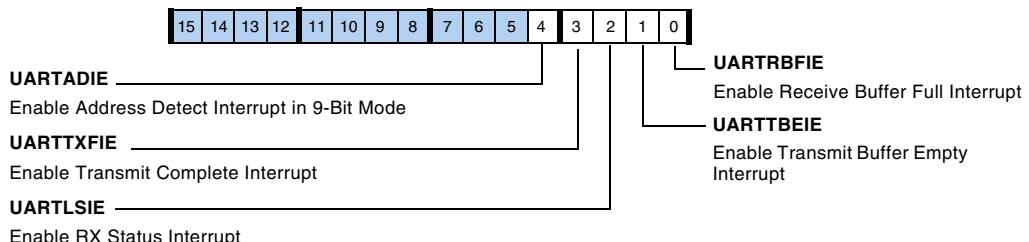


Figure A-138. UART0IER Register

Table A-129. UART0IER Register Bit Descriptions (RW)

Bit	Name	Description
0	UARTRBFIE	Enable Receive Buffer Full Interrupt. 0 = No interrupt 1 = Generate RX interrupt if DR bit in UART_LSR is set
1	UARTTBEIE	Enable Transmit Buffer Empty Interrupt. 0 = No interrupt 1 = Generate TX interrupt if THRE bit in UART_LSR is set
2	UARTLSIE	Enable RX Status Interrupt. 0 = No interrupt 1 = Generate line status interrupt if any of UART_LSR[4-1] is set
3	UARTRXFIE	Enable Transmit Complete Interrupt. 0 = No interrupt 1 = Generate TX interrupt if TEMT bit in UART_LSR is set
4	UARTADIE	Enable Address Detect Interrupt in 9-Bit Mode. 0 = No interrupt 1 = Generate RX interrupt when address is detected in 9-bit mode

Interrupt Identification Registers (UART0IIR, UART0IIRSH)

For legacy reasons, the UART interrupt identification register (UART0IIR, shown in [Figure A-139](#)) 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.

Peripherals Routed Through the DPI

This can be established by globally assigning all UART interrupts to the same interrupt priority. [For more information, see Appendix B, Peripheral Interrupt Control.](#)

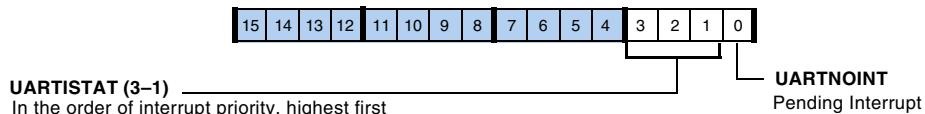


Figure A-139. UART0IIRSH Register

Table A-130. UART0IIRSH Register Bit Descriptions (RO)

Bit	Name	Description
0	UARTNOINT	Pending Interrupt. When UARTNOINT bit cleared it signals that an interrupt is pending. 0 = Interrupt pending 1 = No interrupt pending (default)
3–1 (ROC ¹)	UARTISTAT	In the Order of Interrupt Priority, Highest First. 011 = Receive line status. Read UART_LSR to clear interrupt request. 100 = Address detect. Read RBR to clear interrupt request. 010 = Receive data ready. Read UART_RBR to clear interrupt request. 001 = UART_THR empty. Write UART_THR or read UART_IIR to clear interrupt request, when priority = 4. 000 = UART THR & TSR empty (TEMT = transmit complete). Write UART_THR or read UART_IIR to clear interrupt request, when priority = 5. 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.

1 These bits are read-only in the UARTxLRSW (shadow) register.

There is also a shadow register, `UART0IIRSH`. This register allows programs to read the contents of the corresponding main register without affecting the status of the UART.

Divisor Latch Registers (UART0DLL, UART0DLH)

The bit rate is characterized by the peripheral clock (PCLK) and the 16-bit divisor. The divisor is split into the UART divisor latch low byte register (UART0DLL) and the UART divisor latch high byte register (UART0DLH), both shown in [Figure A-140](#).

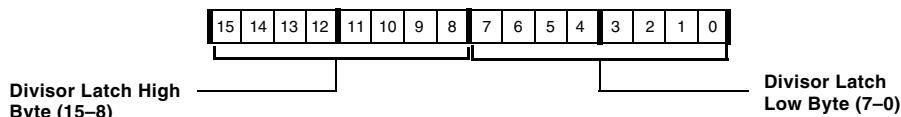


Figure A-140. UART Divisor Latch Registers

Scratch Register (UART0SCR)

This register ([Figure A-141](#)) is used for general-purpose data storage and does not control the UART hardware in any way.

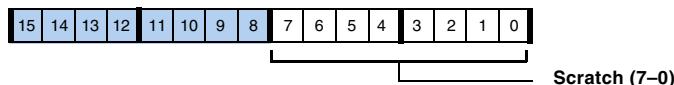


Figure A-141. UART Scratch Registers

Mode Register (UART0MODE)

The UART mode register controls miscellaneous settings as shown in [Figure A-142](#) and described [Table A-131](#).

Peripherals Routed Through the DPI

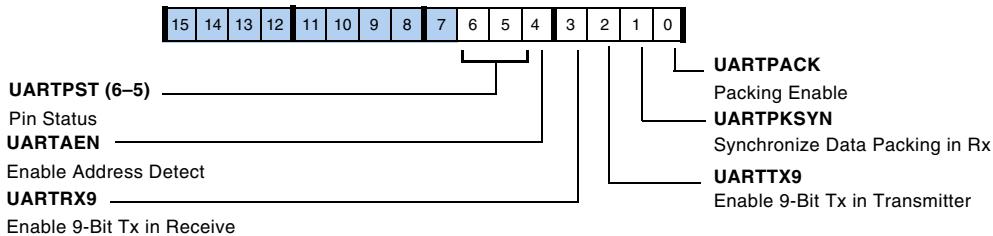


Figure A-142. UART Mode Register

Table A-131. UART Mode Register Bit Descriptions (RW)

Bit	Name	Description
0	UARTPACK	Packing Enable. 0 = No pack 1 = Packing enabled. Consecutive data words (example 0xAB and 0xCD) are packed as 0x00CD 00AB in the receiver, and 0x00CD 00AB is transmitted as two words of 0xAB and 0xCD successively from the transmitter. For more information, see “Data Packing” on page 20-8.
1 (WO)	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	UARTRAEN	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

Table A-131. UART Mode Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
6–5 (RO)	UARTPST	<p>Transmit Pin Status.</p> <p>00 = UART0_TX_O is low 01 = UART0_TX_O is three-stated (default) 10 = UART0_TX_O is three-stated 11 = UART0_TX_O is high</p>

Buffer Control Registers (UART0TXCTL, UART0RXCTL)

Use these registers (described in [Table A-132](#) and [Table A-133](#)) to enable the UART buffers, standard DMA, and DMA chaining.

Table A-132. UART0TXCTL Register Descriptions (RW)

Bit	Name	Description
0	UARTEN	<p>Transmit Buffer Enable.</p> <p>0 = Clear transmit buffer 1 = Enables the transmit buffer (this bit needs to be enabled regardless of core or DMA access)</p>
1	UARTDEN	<p>DMA Enable.</p> <p>0 = Disable DMA 1 = Enable DMA on the specified channel</p>
2	UARTCHEN	<p>Chain Pointer DMA Enable.</p> <p>0 = Disable chained DMA 1 = Enable chained DMA on the specified channel</p>

Peripherals Routed Through the DPI

Table A-133. UART0RXCTL Register Descriptions (RW)

Bit	Name	Description
0	UARTEN	Receive Buffer Enable. 0 = Clears the receive buffer 1 = Enables the receive buffer (this bit needs to be enabled regardless of core or DMA access)
1	UARTDEN	DMA Enable. 0 = Disables DMA 1 = Enables DMA on the specified channel
2	UARTCHEN	Chain Pointer DMA Enable. 0 = Disable chained DMA 1 = Enable chained DMA on the specified channel

DMA Status Registers (UART0TXSTAT, UART0RXSTAT)

These read-only registers (described in [Table A-134](#) and [Table A-135](#)) provide DMA status information.

Table A-134. UART0TXSTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	Reserved	
1	UARTDMASTAT	DMA Status. Provides DMA status. 0 = TX DMA is inactive 1 = TX DMA is active
2	UARTCHSTAT	DMA Chaining Status. Provides DMA chaining status. 0 = TX DMA chain loading is inactive 1 = TX DMA chain loading is active

Table A-135. UART0RXSTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	UARTEERRIRQ	Receive Channel Error Interrupt. 0 = No error interrupt 1 = Error interrupt generated due to receive error (parity/over-run/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

The two wire interface (TWI) registers provide all control and status bits for this peripheral. Status bits can be updated by their respective functional blocks.

Master Internal Time Register (TWIMITR)

The TWIMITR register, shown in [Figure A-143](#) and described in [Table A-136](#), 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:

$$\text{PRESCALE} = f_{\text{PCIK}} / 10 \text{ MHz}$$

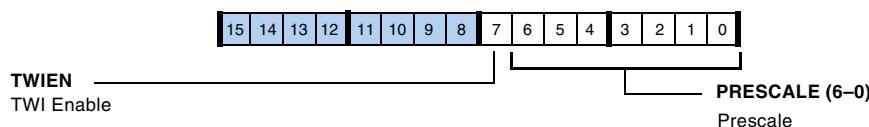


Figure A-143. TWIMITR Register

Peripherals Routed Through the DPI

Table A-136. TWIMITR Register Bit Descriptions (RW)

Bit	Name	Description
0–6	PRESCALE	Prescale. The number of peripheral clock (PCLK) periods used in the generation of one internal time reference. The value of PRESCALE must be set to create an internal time reference with a period of 10 MHz. This is represented as a 7-bit binary value.
7	TWIEN	TWI Enable. This bit must be set for slave or master mode operation. It is recommended that this bit be set at the time PRESCALE is initialized and remain set. This guarantees accurate operation of bus busy detection logic. 0 = Disable TWI 1 = Enable TWI master and slave mode operation

Clock Divider Register (TWIDIV)

During master mode operation, the SCL clock divider register (TWIDIV) shown in [Figure A-144](#) and described in [Table A-137](#) values are used to create the high and low durations of the serial clock (SCL). Serial clock frequencies can vary from 400 KHz to less than 20 KHz. The resolution of the clock generated is 1/10 MHz or 100 ns.

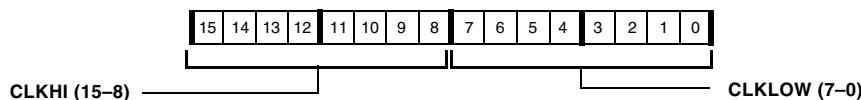


Figure A-144. TWIDIV Register

Table A-137. TWIDIV Register Bit Descriptions (RW)

Bit	Name	Description
7–0	CLKLOW	Clock Low. Number of internal time reference periods the serial clock (TWI_CLK) is held low. Represented as an 8-bit binary value.
15–8	CLKHI	Clock High. Number of internal time reference periods the serial clock (TWI_CLK) waits before a new clock low period begins (assuming a single master). Represented as an 8-bit binary value.

Slave Mode Control Register (TWISCTL)

The TWI slave mode control register (TWISCTL) shown in [Figure A-145](#) and described in [Table A-138](#), controls the logic associated with slave mode operation. Settings in this register do not affect master mode operation and should not be modified to control master mode functionality.

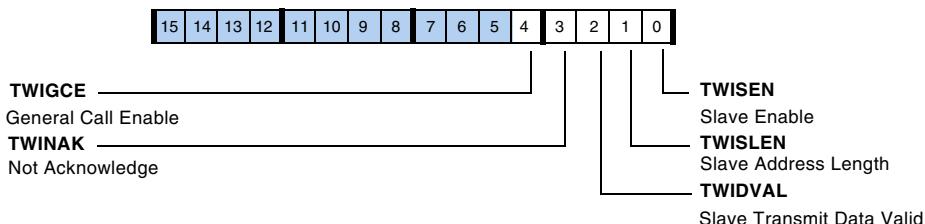


Figure A-145. TWISCTL Register

Table A-138. TWISCTL Register Bit Descriptions (RW)

Bit	Name	Description
0	TWISEN	Slave Enable. 0 = The slave is not enabled. No attempt is made to identify a valid address. If cleared during a valid transfer, clock stretching ceases, the serial data line is released and the current byte is not acknowledged. 1 = The slave is enabled. Enabling slave and master modes of operation concurrently is allowed.
1	TWISLEN	Slave Address Length. 0 = Address is a 7-bit address 1 = Reserved. Setting this bit to 1 causes unpredictable behavior.
2	TWIDVAL	Slave Transmit Data Valid. 0 = Data in the transmit FIFO is for master mode transmits and is not allowed to be used during a slave transmit, and the transmit FIFO is treated as if it is empty. 1 = Data in the transmit FIFO is available for a slave transmission.

Peripherals Routed Through the DPI

Table A-138. TWISCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
3	TWINAK	Not Acknowledged. 0 = Slave receive transfer generates an ACK at the conclusion of a data transfer. 1 = Slave receive transfer generates a data NAK at the conclusion of a data transfer. The slave is still considered to be addressed.
4	TWIGCE	General Call Enable. General call address detection is available only when slave mode is enabled. 0 = General call address matching is not enabled 1 = General call address matching is enabled. Regardless of the selected address length of slave address, a general call slave receive transfer is accepted. All status and interrupt source bits associated with transfers are updated.

Slave Address Register (TWISADDR)

The TWI slave mode address register (TWISADDR, shown in [Figure A-146](#)) holds the slave mode address, which is the valid address that the slave-enabled TWI controller responds to. The TWI controller compares this value with the received address during the addressing phase of a transfer.

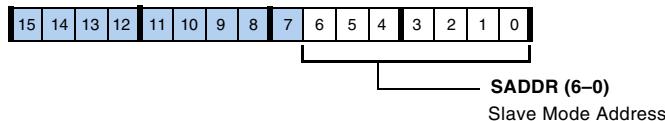


Figure A-146. TWISADDR Register (RW)

Slave Status Register (TWISSTAT)

During and at the conclusion of slave mode transfers, the TWI slave mode status register (shown in [Figure A-147](#)) holds information on the current transfer. Generally slave mode status bits are not associated with the generation of interrupts. Master mode operation does not affect the slave mode status bits.

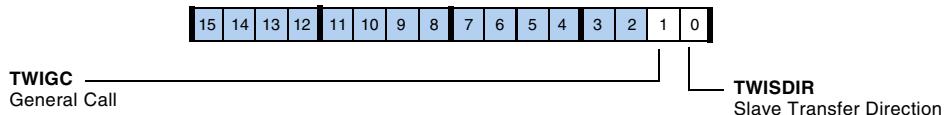


Figure A-147. TWISSTAT Register

Table A-139. TWISSTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	TWISDIR	Slave Transfer Direction. This bit self clears if slave mode is disabled (SEN = 0) 0 = At the time of addressing, the transfer direction was determined to be slave receive. 1 = At the time of addressing, the transfer direction was determined to be slave transmit.
1	TWIGC	General Call. This bit self clears if slave mode is disabled (SEN = 0) 0 = At the time of addressing, the address was not determined to be a general call. 1 = At the time of addressing, the address was determined to be a general call.

Master Control Register (TWIMCTL)

The TWI master mode control register (`TWIMCTL`, shown in [Figure A-148](#) and described in [Table A-140](#)) 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.

Peripherals Routed Through the DPI

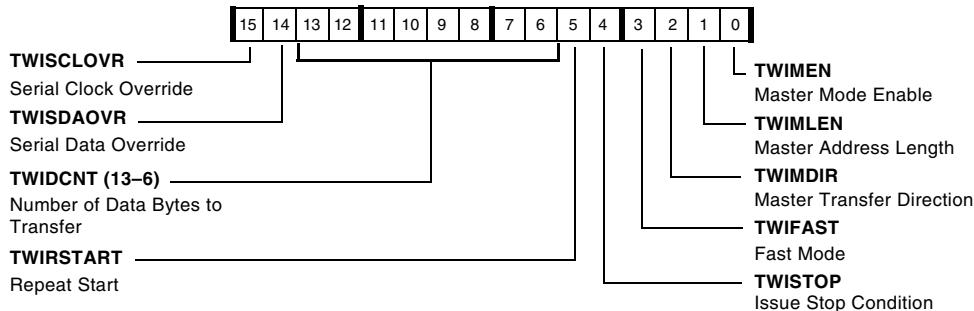


Figure A-148. TWIMCTL Register

Table A-140. TWIMCTL Register Bit Descriptions (RW)

Bit	Name	Description
0	TWIMEN	Master Mode Enable. Clears itself at the completion of a transfer. This includes transfers terminated due to errors. 0 = Master mode functionality is disabled. If MEN is cleared during operation, the transfer is aborted and all logic associated with master mode transfers are reset. Serial data and serial clock (TWI_DATA, TWI_CLOCK) are no longer driven. Write 1-to-clear status bits are not effected. 1 = Master mode functionality is enabled. A START condition is generated if the bus is idle.
1	TWIMLEN	Master Address Length. 0 = Address is 7-bit 1 = Reserved. Setting this bit to one causes unpredictable behavior.
2	TWIMDIR	Master Transfer Direction. 0 = The initiated transfer is master transmit 1 = The initiated transfer is master receive
3	TWIFAST	Fast Mode. 0 = Standard mode timing specifications in use 1 = Fast mode timing specifications in use

Table A-140. TWIMCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
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.
13–6	TWIDCNT	Data Transfer Count. Indicates the number of data bytes to transfer. As each data word is transferred, the data transfer count is decremented. When DCNT is zero, a STOP (or restart condition) is issued. Setting DCNT to 0xFF disables the counter. In this transfer mode, data continues to be transferred until it is concluded by setting the STOP bit.
14	TWISDAOVR	Serial Data (TWI_DATA) Override. For use when direct control of the Serial Data line is required. Normal master and slave mode operation should not require override operation. 0 = Normal serial data operation under the control of the transmit shift register and acknowledge logic 1 = Serial data output is driven to an active “zero” level, overriding all other logic. This state is held until the bit location is cleared.
15	TWISCLOVR	Serial Clock (TWI_Clock) Override. For use when direct control of the serial clock line is required. Normal master and slave mode operation should not require override operation. 0 = Normal serial clock operation under the control of master mode clock generation and slave mode clock stretching logic 1 = Serial clock output is driven to an active “zero” level, overriding all other logic. This state is held until the bit location is cleared.

Master Address Register (TWIMADDR)

During the addressing phase of a transfer, the TWI controller, with its master enabled, transmits the contents of the TWI master mode address register (TWIMADDR, shown in [Figure A-149](#)). 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.

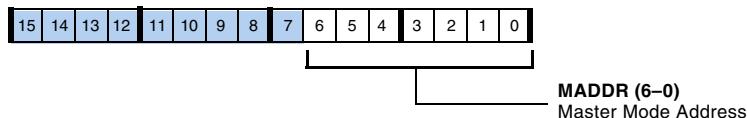


Figure A-149. TWIMADDR Register (RW)

Master Status Register (TWIMSTAT)

The TWI master mode status register (TWIMSTAT, shown in [Figure A-150](#) and described in [Table A-141](#)) holds information during master mode transfers and at their conclusions. Generally, master mode status bits are not directly associated with the generation of interrupts but offer information on the current transfer. Slave mode operation does not affect master mode status bits.

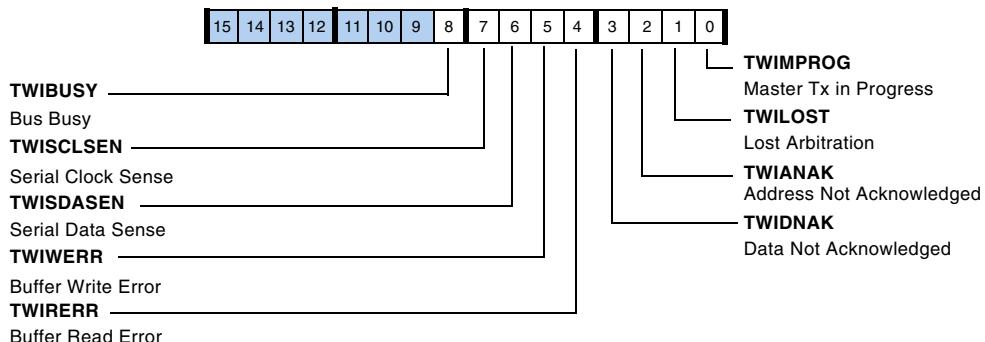


Figure A-150. TWIMSTAT Register

Table A-141. TWIMSTAT Register Bit Descriptions (RO)

Bit	Name	Description
0	TWIMPROG	Master Transfer In Progress. 0 = Currently no transfer is taking place. This can occur once a transfer is complete or while an enabled master is waiting for an idle bus. 1 = A master transfer is in progress.
1 (W1C)	TWILOST	Lost Arbitration. 0 = The current transfer has not lost arbitration with another master. 1 = The current transfer was aborted due to the loss of arbitration with another master.
2 (W1C)	TWIANAK	Address Not Acknowledged. 0 = The current master transfer has not detected a NAK during addressing. 1 = The current master transfer was aborted due to the detection of a NAK during the address phase of the transfer.
3 (W1C)	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.

Peripherals Routed Through the DPI

Table A-141. TWIMSTAT Register Bit Descriptions (RO) (Cont'd)

Bit	Name	Description
4 (W1C)	TWIRERR	Buffer Read Error. 0 = The current master transmit has not detected a buffer read error. 1 = The current master transfer was aborted due to a transmit buffer read error. At the time data was required by the transmit shift register, the buffer was empty.
5 (W1C)	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.
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-151](#) and described in [Table A-142](#)) affects only the FIFO and is not tied in any way with master or slave mode operation.

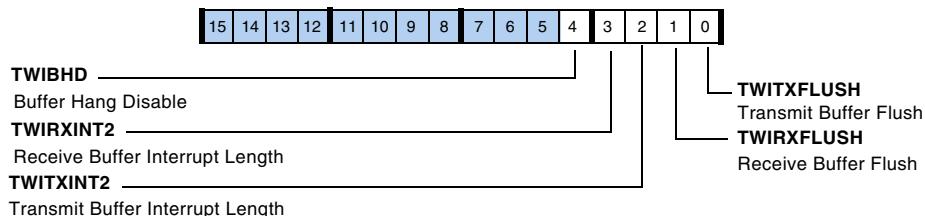


Figure A-151. TWIFIFOCTL Register

Table A-142. TWIFIFOCTL Register Bit Descriptions (RW)

Bit	Name	Description
0	TWITXFLUSH	Transmit Buffer Flush. 0 = Normal operation of the transmit buffer and its status bits 1 = Flush the contents of the transmit buffer and update the status to indicate the buffer is empty. This state is held until this bit is cleared. During an active transmit, the transmit buffer in this state responds as if the transmit buffer is empty.
1	TWIRXFLUSH	Receive Buffer Flush. 0 = Normal operation of the receive buffer and its status bits. 1 = Flush the contents of the receive buffer and update the status to indicate the buffer is empty. This state is held until this bit is cleared. During an active receive the receive buffer in this state responds to the receive logic as if it is full.
2	TWITXINT2	Transmit Buffer Interrupt Length. Determines the rate at which transmit buffer interrupts are generated. Interrupts may be generated with each byte transmitted or after two bytes are transmitted. 0 = An interrupt (TWITXINT) is set when TWITXS indicates one or two bytes in the FIFO are empty (01 or 00). 1 = An interrupt (TWITXINT) is set when TWITXS indicates two bytes in the FIFO are empty (00).

Peripherals Routed Through the DPI

Table A-142. TWIFIFOCTL Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
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 (TWIFIFOSTAT)

The fields in the TWI FIFO status register (TWIFIFOSTAT, shown in [Figure A-152](#) and described in [Table A-143](#)) indicate the state of the FIFO buffers' receive and transmit contents. The FIFO buffers do not discriminate between master data and slave data. By using the status and control bits provided, the FIFO can be managed to allow simultaneous master and slave operation.

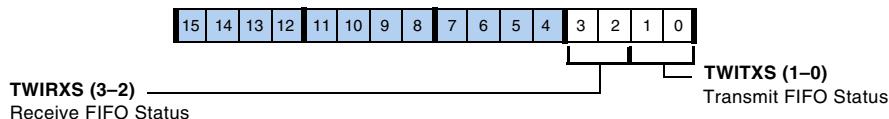


Figure A-152. TWIFIFOSTAT Register

Table A-143. TWIFIFOSTAT Register Bit Descriptions (RO)

Bit	Name	Description
1–0	TWITXS	<p>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.</p> <p>00 = FIFO is empty. Either a single- or double-byte peripheral write of the FIFO goes through immediately.</p> <p>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</p> <p>11 = FIFO is full and contains two bytes of data.</p> <p>10 = Illegal</p>
3–2	TWIRXS	<p>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.</p> <p>00 = FIFO is empty.</p> <p>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.</p> <p>11 = FIFO is full and contains two bytes of data. Either a single- or double-byte peripheral read of the FIFO is allowed.</p> <p>10 = Illegal</p>

Interrupt Latch Register (TWIIRPTL)

The TWI interrupt source register (TWIIRPTL, shown in [Figure A-153](#) and described in [Table A-144](#)) 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.

Peripherals Routed Through the DPI

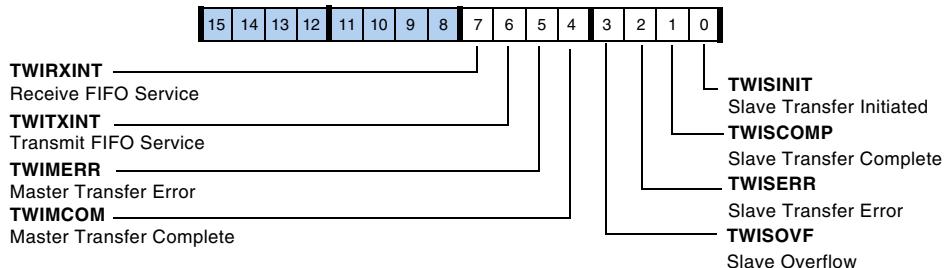


Figure A-153. TWIIRPTL Register

Table A-144. TWIIRPTL Register Bit Descriptions (W1X)

Bit	Name	Description
0	TWISINIT	Slave Transfer Initiated. 0 = A transfer is not in progress. An address match has not occurred since the last time this bit was cleared. 1 = The slave has detected an address match and a transfer has been initiated.
1	TWISCOMP	Slave Transfer Complete. 0 = The completion of a transfer not detected 1 = The transfer is complete and either a stop, or a restart was detected.
2	TWISERR	Slave Transfer Error. 0 = No errors detected 1 = An error has occurred. A restart or stop condition has occurred during the data receive phase of a transfer.
3	TWISOVF	Slave Overflow. 0 = No overflow detected 1 = The slave transfer complete (TWISCOMP) was set at the time a subsequent transfer has acknowledged an address phase. The transfer continues, however, it may be difficult to delineate data of one transfer from another.
4	TWIMCOM	Master Transfer Complete. 0 = The completion of a transfer not detected 1 = The initiated master transfer is complete. In the absence of a repeat start, the bus is released.

Table A-144. TWIIRPTL Register Bit Descriptions (W1X) (Cont'd)

Bit	Name	Description
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).
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.
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.

Interrupt Enable Register (TWIIMASK)

The TWI interrupt mask register (TWIIMASK, shown in [Figure A-154](#) and described in [Table A-145](#)) enables interrupt sources to assert the interrupt output. Each enable bit corresponds with one interrupt latch bit in the TWI interrupt latch register (TWIIRPTL). Reading and writing the TWIIMASK register does not affect the contents of the TWIIRPTL register.

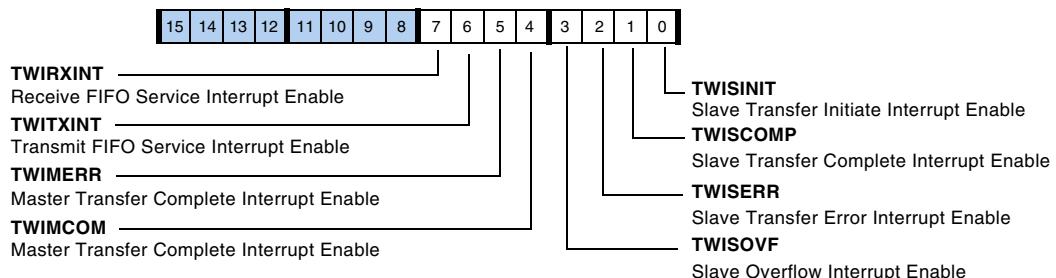


Figure A-154. TWIIMASK Register

Peripherals Routed Through the DPI

Table A-145. TWIIMASK Register Bit Descriptions (RW)

Bit	Name	Description
0	TWISINIT	Slave Transfer Initiate Interrupt Enable. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
1	TWISCOMP	Slave Transfer Complete Interrupt. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
2	TWISERR	Slave Transfer Error Interrupt. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
3	TWISOVF	Slave Overflow Interrupt Enable. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
4	TWIMCOM	Master Transfer Complete Interrupt Enable. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
5	TWIMERR	Master Transfer Error Interrupt Enable. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
6	TWITXINT	Transmit FIFO Service Interrupt Enable. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.
7	TWIRXINT	Receive FIFO Service Interrupt Enable. 0 = The corresponding interrupt source is prevented from asserting the interrupt output. 1 = The corresponding interrupt source asserts the interrupt output.

Peripheral Timer Registers

The timer peripheral module provides general-purpose timer functionality. It consists of three identical timer units. Each timer has memory-mapped registers. They are described in the following sections.

Read-Modify-Write Timer Control Register

For the Timer global control register, the traditional read-modify-write operations to disable a timer have changed. The action is to directly write which simplifies timer enable/disable and can be accomplished with fewer instructions. Example:

Instead of:

```
ustat3=dm(TMCTL);           /* Timer Control Register */
bit set ustat3 TIM1DIS;    /* disables timer 1 */
dm(TMCTL)=ustat3;
```

Use:

```
ustat3 = TIM1DIS;
dm(TMCTL)=ustat3;
```

Writes to the enable and disable bit-pair for a timer works as follows.

$\text{TIMxDIS} = 0, \text{TIMxEN} = 0$ – No action

$\text{TIMxDIS} = 0, \text{TIMxEN} = 1$ – Enable the timer

$\text{TIMxDIS} = 1, \text{TIMxEN} = x$ – Disable the timer

For reads, the interpretation is as follows.

$\text{TIM1DIS} = 0, \text{TIMxEN} = 0$ – Timer is disabled

$\text{TIM1DIS} = 1, \text{TIMxEN} = 1$ – Timer is enabled

Any other read combination is not possible. Read of the `TMCTL` register returns the enable status on both the enable and disable bits.

Timer Configuration Registers (TMxCTL)

All timer clocks are gated off when the specific timer's configuration register is set to zero at system reset or subsequently reset by user programs. These registers are shown in [Figure A-155](#).

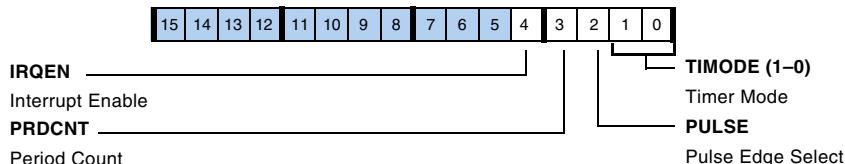


Figure A-155. TMxCTL Register

Table A-146. TMxCTL Register Bit Descriptions (RW)

Bit	Name	Description
1–0	TIMODE	Timer Mode. 00 = Reset 01 = PWM_OUT mode (TIMODEPWM) 10 = WDTH_CAP mode (TIMODEW) 11 = EXT_CLK mode (TIMODEEXT)
2	PULSE	Pulse Edge Select. 1 = Positive active pulse 0 = Negative active pulse
3	PRDCNT	Period Count. 1 = Count to end of period 0 = Count to end of width
4	IRQEN	Interrupt Enable. 1 = Enable 0 = Disable

Timer Status Registers (TMxSTAT)

The global status registers TMxSTAT are shown in [Figure A-156](#). Status bits are sticky and require a write-one to clear operation. During a status register read access, all reserved or unused bits return a zero. Each timer generates a unique processor interrupt request signal, TIMxIRQ.

A common status register latches these interrupts. Interrupt bits are sticky and must be cleared to assure that the interrupt is not reissued.

Each timer is provided with its own sticky status register TIMxEN bit. To enable or disable an individual timer, the TIMxEN bit is set or cleared. For example, writing a one to bit 8 sets the TIM0EN bit; writing a one to bit 9 clears it. Writing a one to both bit 8 and bit 9 clears TIM0EN. Reading the status register returns the TIM0EN state on both bit 8 and bit 9. The remaining TIMxEN bits operate similarly.

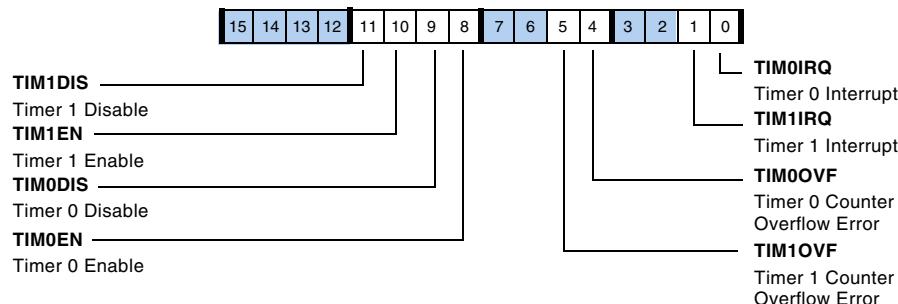


Figure A-156. TMxSTAT Register

Table A-147. TMxSTAT Register Bit Descriptions (RW)

Bit	Name	Description
0 (W1C)	TIM0IRQ Timer 0 Interrupt Latch	Also an output
1 (W1C)	TIM1IRQ Timer 1 Interrupt Latch	Also an output
3–2	Reserved	
4 (RO)	TIM0OVF Timer 0 Overflow/Error	Also an output

Peripherals Routed Through the DPI

Table A-147. TMxSTAT Register Bit Descriptions (RW) (Cont'd)

Bit	Name	Description
5 (RO)	TIM1OVF Timer 1 Overflow/Error	Also an output
7–6	Reserved	
8	TIM0EN Timer 0 Enable	Enable timer 0
9 (W1C)	TIM0DIS Timer 0 Disable	Disable timer 0
10	TIM1EN Timer 1 Enable	Enable timer 1

Register Listing

This section lists all available memory mapped IOP registers including the address and reset values.

Register Mnemonic	Address	Description	Reset
Power Management Register			
PMCTL	0x2000	Power Management Control	Hardware dependent
PMCTL1	0x2001	Power Management Control 1	0
Miscellaneous Registers			
RUNRSTCTL	0x2100	Running Reset Control	0x0
ROMID	0x20FF	ROM Identification	Hardware dependent
Asynchronous Memory Interface Registers			
EPCTL	0x1801	External Port Global Control	0xF0
AMICTL0	0x1804	AMI Control Register for Bank 1	0x0
AMICTL1	0x1805	AMI Control Register for Bank 2	0x0
AMICTL2	0x1806	AMI Control Register for Bank 3	0x0
AMICTL3	0x1807	AMI Control Register for Bank 4	0x0
AMISTAT	0x180A	AMI Status	0x1
DMA Address Registers			
DMAC0	0x180B	External Port DMA CH 0 Control	0x0
EIEP0	0x1820	External Port CH 0 DMA External Index Address	0x0
EMEP0	0x1821	External Port CH 0 DMA External Modifier	0x0
ECEP0	0x1822	External Port CH 0 DMA External Count	0x0
IIEP0	0x1823	External Port CH 0 DMA Internal Index Address	0x0
IMEP0	0x1824	External Port CH 0 DMA Internal Modifier	0x0
ICEP0	0x1825	External Port CH 0 DMA Internal Count	0x0

Register Listing

Register Mnemonic	Address	Description	Reset
CPEP0	0x1826	External Port CH 0 DMA Chain Pointer	0x0
EBEP0	0x1827	External Port CH 0 DMA External Base Address	0x0
TPEP0	0x1828	External Port CH 0 DMA TAP Pointer	0x0
ELEP0	0x1829	External Port CH 0 DMA External Length	0x0
EPTC0	0x182B	External Port CH 0 DMA Delay Line TAP Count	0x0
DFEP0	0x182C	External Port CH 0 DMA Data FIFO	0x0
DMAC1	0x180C	External Port DMA CH 1 Control	0x0
EIEP1	0x1830	External Port CH 1 DMA External Index Address	0x0
EMEP1	0x1831	External Port CH 1 DMA External Modifier	0x0
ECEP1	0x1832	External Port CH 1 DMA External Count	0x0
IIEP1	0x1833	External Port CH 1 DMA Internal Index Address	0x0
IMEP1	0x1834	External Port CH 1 DMA Internal Modifier	0x0
ICEP1	0x1835	External Port CH 1 DMA Internal Count	0x0
CPEP1	0x1836	External Port CH 1 DMA Chain Pointer	0x0
EBEP1	0x1837	External Port CH 1 DMA External Base Address	0x0
ELEP1	0x1839	External Port CH 1 DMA External Length	0x0
EPTC1	0x183B	External Port CH 1 DMA Delay Line TAP Count	0x0
DFEP1	0x183C	External Port CH 1 DMA Data FIFO	0x0
DDR2 Controller Registers			
DDR2CTL0	0x1812	DDR2 Control 0	0x1800 0018
DDR2CTL1	0x1813	DDR2 Control 1	0x9452 3466
DDR2CTL2	0x1814	DDR2 Control 2	0x422
DDR2CTL3	0x1815	DDR2 Control 3	0x4780
DDR2CTL4	0x1816	DDR2 Control 4	0x8000
DDR2CTL5	0x1817	DDR2 Control 5	0xC000
DDR2RRC	0x181D	DDR2 Refresh Rate	0x2800614
DDR2STAT0	0x181E	DDR2 Status 0	0x8

Register Mnemonic	Address	Description	Reset
DDR2STAT1	0x1840	DDR2 Status 1	0x0
DDR2PADCTL0	0x1841	DDR2 Pad Control 0	0x200 0000
DDR2PADCTL1	0x1842	DDR2 Pad Control 1	0x8020
DLL0CTL1	0x1851	DLL0 Control Register 1	0x0
DLL0STAT0	0x1853	DLL0 Status Register 1	0x0
DLL1CTL1	0x1856	DLL1 Control Register 1	0x0
DLL1STAT0	0x1858	DLL1 Status Register 1	0x0
SDRAM Registers			
SDCTL	0x1800	SDRAM Control	0x0102000A
SDRRC	0x1802	SDRAM Refresh Count	0x3081A
SDSTAT0	0x1803	SDRAM Status	0x8
SDSTAT1	0x1804	SDRAM Status	0x8
Global Serial Port Register			
SPERRSTAT	0x2300	Global SPORT Error Status	0x0
Serial Port Error Control Registers			
SPERRCTL0	0xC18	SPORT0 Error Control	0x0
SPERRCTL1	0xC19	SPORT1 Error Control	0x0
SPERRCTL2	0x418	SPORT2 Error Control	0x0
SPERRCTL3	0x419	SPORT3 Error Control	0x0
SPERRCTL4	0x818	SPORT4 Error Control	0x0
SPERRCTL5	0x819	SPORT5 Error Control	0x0
SPERRCTL6	0x4818	SPORT6 Error Control	0x0
SPERRCTL7	0x4819	SPORT7 Error Control	0x0
Serial Port 0 and 1 Registers			
SPCTL0	0xC00	SPORT 0 Control Register	0x0000 0000
SPCTL1	0xC01	SPORT 1 Control Register	0x0000 0000
SPCTLN0	0xC1A	SPORT 0 Control Register 2	0x0000 0000

Register Listing

Register Mnemonic	Address	Description	Reset
SPCTLN1	0xC1B	SPORT 1 Control Register 2	0x0000 0000
DIV0	0xC02	SPORT 0 Divisor for TX/RX SCLK0 and FS0	0x0
DIV1	0xC03	SPORT 1 Divisor for TX/RX SCLK1 and FS1	0x0
SPMCTL0	0xC04	SPORT 0 TDM Control Register	0x0
SPMCTL1	0xC17	SPORT 1 TDM Control Register	0x0
MT0CS0	0xC05	SPORT 0 TDM TX Select, CH31–0	0x0
MT0CS1	0xC06	SPORT 0 TDM TX Select, CH63–32	0x0
MT0CS2	0xC07	SPORT 0 TDM TX Select, CH95–64	0x0
MT0CS3	0xC08	SPORT 0 TDM TX Select, CH127–96	0x0
MR1CS0	0xC09	SPORT 1 TDM RX Select, CH31–0	0x0
MR1CS1	0xC0A	SPORT 1 TDM RX Select, CH63–32	0x0
MR1CS2	0xC0B	SPORT 1 TDM RX Select, CH95–64	0x0
MR1CS3	0xC0C	SPORT 1 TDM RX Select, CH127–96	0x0
MT0CCS0	0xC0D	SPORT 0 TDM TX Comand Select, CH31–0	0x0
MT0CCS1	0xC0E	SPORT 0 TDM TX Comand Select, CH63–32	0x0
MT0CCS2	0xC0F	SPORT 0 TDM TX Comand Select, CH95–64	0x0
MT0CCS3	0xC10	SPORT 0 TDM TX Comand Select, CH127–96	0x0
MR1CCS0	0xC11	SPORT 1 TDM RX Comand Select, CH31–0	0x0
MR1CCS1	0xC12	SPORT 1 TDM RX Comand Select, CH63–32	0x0
MR1CCS2	0xC13	SPORT 1 TDM RX Comand Select, CH95–64	0x0
MR1CCS3	0xC14	SPORT 1 TDM RX Comand Select, CH127–96	0x0
IISP0A	0xC40	Internal Memory DMA Address	0x0
IMSP0A	0xC41	Internal Memory DMA Access Modifier	0x0
CSP0A	0xC42	Contains Number of DMA Transfers Remaining	0x0
CPSP0A	0xC43	Points to Next DMA Parameters	0x0
IISP0B	0xC44	Internal Memory DMA Address	0x0
IMSP0B	0xC45	Internal Memory DMA Access Modifier	0x0

Register Mnemonic	Address	Description	Reset
CSP0B	0xC46	Contains Number of DMA Transfers Remaining	0x0
CPSP0B	0xC47	Points to Next DMA Parameters	0x0
IISP1A	0xC48	Internal Memory DMA Address	0x0
IMSP1A	0xC49	Internal Memory DMA Access Modifier	0x0
CSP1A	0xC4A	Contains Number of DMA Transfers Remaining	0x0
CPSP1A	0xC4B	Points to Next DMA Parameters	0x0
IISP1B	0xC4C	Internal Memory DMA Address	0x0
IMSP1B	0xC4D	Internal Memory DMA Access Modifier	0x0
CSP1B	0xC4E	Contains Number of DMA Transfers Remaining	0x0
CPSP1B	0xC4F	Points to Next DMA Parameters	0x0
TXSP0A	0xC60	SPORT 0A Transmit Data	0x0
RXSP0A	0xC61	SPORT 0A Receive Data	0x0
TXSP0B	0xC62	SPORT 0B Transmit Data	0x0
RXSP0B	0xC63	SPORT 0B Receive Data	0x0
TXSP1A	0xC64	SPORT 1A Transmit Data	0x0
RXSP1A	0xC65	SPORT 1A Receive Data	0x0
TXSP1B	0xC66	SPORT 1B Transmit Data	0x0
RXSP1B	0xC67	SPORT 1B Receive Data	0x0
Serial Port 2 and 3 Registers			
SPCTL2	0x400	SPORT 2 Control	0x0000 0000
SPCTL3	0x401	SPORT 3 Control	0x0000 0000
SPCTLN2	0x41A	SPORT 2 Control Register 2	0x0000 0000
SPCTLN3	0x41B	SPORT 3 Control Register 2	0x0000 0000
DIV2	0x402	SPORT 2 Divisor for TX/RX SCLK2 and FS2	0x0
DIV3	0x403	SPORT 3 Divisor for TX/RX SCLK3 and FS3	0x0
SPMCTL2	0x404	SPORTs 2 TDM Control	0x0
MT2CS0	0x405	SPORT 2 TDM TX Select, CH31–0	0x0

Register Listing

Register Mnemonic	Address	Description	Reset
MT2CS1	0x406	SPORT 2 TDM TX Select, CH63–32	0x0
MT2CS2	0x407	SPORT 2 TDM TX Select, CH95–64	0x0
MT2CS3	0x408	SPORT 2 TDM TX Select, CH127–96	0x0
MR3CS0	0x409	SPORT 3 TDM RX Select, CH31–0	0x0
MR3CS1	0x40A	SPORT 3 TDM RX Select, CH63–32	0x0
MR3CS2	0x40B	SPORT 3 TDM RX Select, CH95–64	0x0
MR3CS3	0x40C	SPORT 3 TDM RX Select, CH127–96	0x0
MT2CCS0	0x40D	SPORT 2 TDM TX Compand Select, CH31–0	0x0
MT2CCS1	0x40E	SPORT 2 TDM TX Compand Select, CH63–32	0x0
MT2CCS2	0x40F	SPORT 2 TDM TX Compand Select, CH95–64	0x0
MT2CCS3	0x410	SPORT 2 TDM TX Compand Select, CH127–96	0x0
MR3CCS0	0x411	SPORT 3 TDM RX Compand Select, CH31–0	0x0
MR3CCS1	0x412	SPORT 3 TDM RX Compand Select, CH63–32	0x0
MR3CCS2	0x413	SPORT 3 TDM RX Compand Select, CH95–64	0x0
MR3CCS3	0x414	SPORT 3 TDM RX Compand Select, CH127–96	0x0
SPMCTL3	0x417	SPORT 3 TDM Control	0x0
IISP2A	0x440	Internal Memory DMA Address	0x0
IMSP2A	0x441	Internal Memory DMA Access Modifier	0x0
CSP2A	0x442	Contains Number of DMA Transfers Remaining	0x0
CPSP2A	0x443	Points to Next DMA Parameters	0x0
IISP2B	0x444	Internal Memory DMA Address	0x0
IMSP2B	0x445	Internal Memory DMA Access Modifier	0x0
CSP2B	0x446	Contains Number of DMA Transfers Remaining	0x0
CPSP2B	0x447	Points to Next DMA Parameters	0x0
IISP3A	0x448	Internal Memory DMA Address	0x0
IMSP3A	0x449	Internal Memory DMA Access Modifier	0x0
CSP3A	0x44A	Contains Number of DMA Transfers Remaining	0x0

Register Mnemonic	Address	Description	Reset
CPSP3A	0x44B	Points to Next DMA Parameters	0x0
IISP3B	0x44C	Internal Memory DMA Address	0x0
IMSP3B	0x44D	Internal Memory DMA Access Modifier	0x0
CSP3B	0x44E	Contains Number of DMA Transfers Remaining	0x0
CPSP3B	0x44F	Points to Next DMA Parameters	0x0
TXSP2A	0x460	SPORT 2A Transmit Data	0x0
RXSP2A	0x461	SPORT 2A Receive Data	0x0
TXSP2B	0x462	SPORT 2B Transmit Data	0x0
RXSP2B	0x463	SPORT 2B Receive Data	0x0
TXSP3A	0x464	SPORT 3A Transmit Data	0x0
RXSP3A	0x465	SPORT 3A Receive Data	0x0
TXSP3B	0x466	SPORT 3B Transmit Data	0x0
RXSP3B	0x467	SPORT 3B Receive Data	0x0
Serial Port 4 and 5 Registers			
SPCTL4	0x800	SPORT 4 Control	0x0000 0000
SPCTL5	0x801	SPORT 5 Control	0x0000 0000
SPCTLN4	0x81A	SPORT 4 Control Register 2	0x0000 0000
SPCTLN5	0x819	SPORT 5 Control Register 2	0x0000 0000
DIV4	0x802	SPORT 4 Divisor for TX/RX SCLK4 and FS4	0x0
DIV5	0x803	SPORT 5 Divisor for TX/RX SCLK5 and FS5	0x0
SPMCTL4	0x804	SPORT 4 TDM Control	0x0
MT4CS0	0x805	SPORT 4 TDM TX Select, CH31–0	0x0
MT4CS1	0x806	SPORT 4 TDM TX Select, CH63–32	0x0
MT4CS2	0x807	SPORT 4 TDM TX Select, CH95–64	0x0
MT4CS3	0x808	SPORT 4 TDM TX Select, CH127–96	0x0
MR5CS0	0x809	SPORT 5 TDM RX Select, CH31–0	0x0
MR5CS1	0x80A	SPORT 5 TDM RX Select, CH63–32	0x0

Register Listing

Register Mnemonic	Address	Description	Reset
MR5CS2	0x80B	SPORT 5 TDM RX Select, CH95–64	0x0
MR5CS3	0x80C	SPORT 5 TDM RX Select, CH127–96	0x0
MT4CCS0	0x80D	SPORT 4 TDM TX Comand Select, CH31–0	0x0
MT4CCS1	0x80E	SPORT 4 TDM TX Comand Select, CH63–32	0x0
MT4CCS2	0x80F	SPORT 4 TDM TX Comand Select, CH95–64	0x0
MT4CCS3	0x810	SPORT 4 TDM TX Comand Select, CH127–96	0x0
MR5CCS0	0x811	SPORT 5 TDM RX Comand Select, CH31–0	0x0
MR5CCS1	0x812	SPORT 5 TDM RX Comand Select, CH63–32	0x0
MR5CCS2	0x813	SPORT 5 TDM RX Comand Select, CH95–64	0x0
MR5CCS3	0x814	SPORT 5 TDM RX Comand Select, CH127–96	0x0
SPMCTL5	0x817	SPORT 5 TDM Control	0x0
IISP4A	0x840	Internal Memory DMA Address	0x0
IMSP4A	0x841	Internal Memory DMA Access Modifier	0x0
CSP4A	0x842	Contains Number of DMA Transfers Remaining	0x0
CPSP4A	0x843	Points to Next DMA Parameters	0x0
IISP4B	0x844	Internal Memory DMA Address	0x0
IMSP4B	0x845	Internal Memory DMA Access Modifier	0x0
CSP4B	0x846	Contains Number of DMA Transfers Remaining	0x0
CPSP4B	0x847	Points to Next DMA Parameters	0x0
IISP5A	0x848	Internal Memory DMA Address	0x0
IMSP5A	0x849	Internal Memory DMA Access Modifier	0x0
CSP5A	0x84A	Contains Number of DMA Transfers Remaining	0x0
CPSP5A	0x84B	Points to Next DMA Parameters	0x0
IISP5B	0x84C	Internal Memory DMA Address	0x0
IMSP5B	0x84D	Internal Memory DMA Access Modifier	0x0
CSP5B	0x84E	Contains Number of DMA Transfers Remaining	0x0
CPSP5B	0x84F	Points to Next DMA Parameters	0x0

Register Mnemonic	Address	Description	Reset
TXSP4A	0x860	SPORT 4A Transmit Data	0x0
RXSP4A	0x861	SPORT 4A Receive Data	0x0
TXSP4B	0x862	SPORT 4B Transmit Data	0x0
RXSP4B	0x863	SPORT 4B Receive Data	0x0
TXSP5A	0x864	SPORT 5A Transmit Data	0x0
RXSP5A	0x865	SPORT 5A Receive Data	0x0
TXSP5B	0x866	SPORT 5B Transmit Data	0x0
RXSP5B	0x867	SPORT 5B Receive Data	0x0
SPORT 6 and 7 Registers			
SPCTL6	0x4800	SPORT 6 Control	0x0000 0000
SPCTL7	0x4801	SPORT 7 Control	0x0000 0000
SPCTLN6	0x481A	SPORT 6 Control Register 2	0x0000 0000
SPCTLN7	0x481B	SPORT 7 Control Register 2	0x0000 0000
DIV6	0x4802	SPORT 6 Divisor for TX/RX SCLK6 and FS6	0x0
DIV7	0x4803	SPORT 7 Divisor for TX/RX SCLK7 and FS7	0x0
SPMCTL6	0x4804	SPORT 6 TDM Control	0x0
MT6CS0	0x4805	SPORT 6 TDM TX Select, CH31–0	0x0
MT6CS1	0x4806	SPORT 6 TDM TX Select, CH63–32	0x0
MT6CS2	0x4807	SPORT 6 TDM TX Select, CH95–64	0x0
MT6CS3	0x4808	SPORT 6 TDM TX Select, CH127–96	0x0
MR7CS0	0x4809	SPORT 7 TDM RX Select, CH31–0	0x0
MR7CS1	0x480A	SPORT 7 TDM RX Select, CH63–32	0x0
MR7CS2	0x480B	SPORT 7 TDM RX Select, CH95–64	0x0
MR7CS3	0x480C	SPORT 7 TDM RX Select, CH127–96	0x0
MT6CCS0	0x480D	SPORT 6 TDM TX Companad Select, CH31–0	0x0
MT6CCS1	0x480E	SPORT 6 TDM TX Companad Select, CH63–32	0x0
MT6CCS2	0x480F	SPORT 6 TDM TX Companad Select, CH95–64	0x0

Register Listing

Register Mnemonic	Address	Description	Reset
MT6CCS3	0x4810	SPORT 6 TDM TX CompaND Select, CH127–96	0x0
MR7CCS0	0x4811	SPORT 7 TDM RX CompaND Select, CH31–0	0x0
MR7CCS1	0x4812	SPORT 7 TDM RX CompaND Select, CH63–32	0x0
MR7CCS2	0x4813	SPORT 7 TDM RX CompaND Select, CH95–64	0x0
MR7CCS3	0x4814	SPORT 7 TDM RX CompaND Select, CH127–96	0x0
SPMCTL7	0x4817	SPORT 7 TDM Control	0x0
IISP6A	0x4840	Internal Memory DMA Address	0x0
IMSP6A	0x4841	Internal Memory DMA Access Modifier	0x0
CSP6A	0x4842	Contains Number of DMA Transfers Remaining	0x0
CPSP6A	0x4843	Points to Next DMA Parameters	0x0
IISP6B	0x4844	Internal Memory DMA Address	0x0
IMSP6B	0x4845	Internal Memory DMA Access Modifier	0x0
CSP6B	0x4846	Contains Number of DMA Transfers Remaining	0x0
CPSP6B	0x4847	Points to Next DMA Parameters	0x0
IISP7A	0x4848	Internal Memory DMA Address	0x0
IMSP7A	0x4849	Internal Memory DMA Access Modifier	0x0
CSP7A	0x484A	Contains Number of DMA Transfers Remaining	0x0
CPSP7A	0x484B	Points to Next DMA Parameters	0x0
IISP7B	0x484C	Internal Memory DMA Address	0x0
IMSP7B	0x484D	Internal Memory DMA Access Modifier	0x0
CSP7B	0x484E	Contains Number of DMA Transfers Remaining	0x0
CPSP7B	0x484F	Points to Next DMA Parameters	0x0
TXSP6A	0x4860	SPORT 6A Transmit Data	0x0
RXSP6A	0x4861	SPORT 6A Receive Data	0x0
TXSP6B	0x4862	SPORT 6B Transmit Data	0x0
RXSP6B	0x4863	SPORT 6B Receive Data	0x0
TXSP7A	0x4864	SPORT 7A Transmit Data	0x0

Register Mnemonic	Address	Description	Reset
RXSP7A	0x4865	SPORT 7A Receive Data	0x0
TXSP7B	0x4866	SPORT 7B Transmit Data	0x0
RXSP7B	0x4867	SPORT 7B Receive Data	0x0
SPI Registers			
SPICTL	0x1000	SPI Control	0x0400
SPIFLG	0x1001	SPI Flag	0x0F80
SPISTAT	0x1002	SPI Status	0x01
TXSPI	0x1003	SPI Transmit Data	0x0
RXSPI	0x1004	SPI Receive Data	0x0
SPIBAUD	0x1005	SPI Baud Setup	0x0
RXSPI_SHADOW	0x1006	SPI Receive Data Shadow	0x0
IISPI	0x1080	Internal Memory DMA Address	0x0
IMSPI	0x1081	Internal Memory DMA Access Modifier	0x0
CSPI	0x1082	Contains Number of DMA Transfers Remaining	0x0
CPSPI	0x1083	Points to Next DMA Parameters	0x0
SPIDMAC	0x1084	SPI DMA Control	0x0
SPIB Registers			
SPICTLB	0x2800	SPIB Control	0x0400
SPIFLGB	0x2801	SPIB Flag	0x0F00
SPISTATB	0x2802	SPIB Status	0x01
TXSPIB	0x2803	SPIB Transmit Data	0x0
RXSPIB	0x2804	SPIB Receive Data	0x0
SPIBAUDB	0x2805	SPIB Baud Setup	0x0
RXSPIB_SHADOW	0x2806	SPIB Receive Data Shadow	0x0
IISPIB	0x2880	Internal Memory DMA Address	0x0
IMSPIB	0x2881	Internal Memory DMA Access Modifier	0x0
CSPIB	0x2882	Contains Number of DMA Transfers Remaining	0x0

Register Listing

Register Mnemonic	Address	Description	Reset
CPSPIB	0x2883	Points to Next DMA Parameters	0x0
SPIDMACB	0x2884	SPIB DMA Control	0x0
Timer Registers			
TM0STAT	0x1400	GP Timer 0 Status	0x0
TM0CTL	0x1401	GP Timer 0 Control	0x0
TM0CNT	0x1402	GP Timer 0 Count	0x0
TM0PRD	0x1403	GP Timer 0 Period	0x0
TM0W	0x1404	GP Timer 0 Width	0x0
TM1STAT	0x1408	GP Timer 1 Status	0x0
TM1CTL	0x1409	GP Timer 1 Control	0x0
TM1CNT	0x140A	GP Timer 1 Count	0x0
TM1PRD	0x140B	GP Timer 1 Period	0x0
TM1W	0x140C	GP Timer 1 Width	0x0
SRU DAI Routing Registers			
SRU_CLK0	0x2430	SRU Clock Control 0	0x2526 30C2
SRU_CLK1	0x2431	SRU Clock Control 1	0x3DEF 7BDE
SRU_CLK2	0x2432	SRU Clock Control 2	0x3DEF 7BDE
SRU_CLK3	0x2433	SRU Clock Control 3	0x3DEF 7BDE
SRU_CLK4	0x2434	SRU Clock Control 4	0x3DEF 7BDE
SRU_CLK5	0x2435	SRU Clock Control 5	0x3DEF 7BDE
SRU_DAT0	0x2440	SRU Data Control 0	0x0814 4040
SRU_DAT1	0x2441	SRU Data Control 1	0x0F38 B289
SRU_DAT2	0x2442	SRU Data Control 2	0x0000 0450
SRU_DAT3	0x2443	SRU Data Control 3	0x0
SRU_DAT4	0x2444	SRU Data Control 4	0x0
SRU_DAT5	0x2445	SRU Data Control 5	0x0
SRU_DAT6	0x2446	SRU Data Control 6	0x00FB EFBE

Register Mnemonic	Address	Description	Reset
SRU_FS0	0x2450	SRU FS Control 0	0x2736 B4E3
SRU_FS1	0x2451	SRU FS Control 1	0x3DEF 7BDE
SRU_FS2	0x2452	SRU FS Control 2	0x3DEF 7BDE
SRU_FS3	0x2453	SRU FS Control 3	0x1EF7BDE
SRU_FS4	0x2454	SRU FS Control 4	0x3DE
SRU_PIN0	0x2460	SRU Pin Control 0	0x04C8 0A94
SRU_PIN1	0x2461	SRU Pin Control 1	0x04E8 4B96
SRU_PIN2	0x2462	SRU Pin Control 2	0x0366 8C98
SRU_PIN3	0x2463	SRU Pin Control 3	0x03A7 14A3
SRU_PIN4	0x2464	SRU Pin Control 4	0x0569 4F9E
SRU_EXT_MISCA	0x2470	SRU External Misc. A Control	0x3DEF 7BDE
SRU_EXT_MISCB	0x2471	SRU External Misc. B Control	0x3DEF 7BDE
SRU_PBEN0	0x2478	SRU Pin Enable 0	0x0E24 82CA
SRU_PBEN1	0x2479	SRU Pin Enable 1	0x1348 D30F
SRU_PBEN2	0x247A	SRU Pin Enable 2	0x1A55 45D6
SRU_PBEN3	0x247B	SRU Pin Enable 3	0x1D71 F79B
SRU2 DPI Routing Registers			
SRU2_INPUT0	0x1C00	SRU2 Input Signal Routing 0	0x0002 1462
SRU2_INPUT1	0x1C01	SRU2 Input Signal Routing 1	0x1AC0 2C00
SRU2_INPUT2	0x1C02	SRU2 Input Signal Routing 2	0x0
SRU2_INPUT3	0x1C03	SRU2 Input Signal Routing 3	0x0
SRU2_INPUT4	0x1C04	SRU2 Input Signal Routing 4	0x0
SRU2_INPUT5	0x1C05	SRU2 Input Signal Routing 5	0x0
SRU2_PIN0	0x1C10	SRU2 Pin Assignment 0	0x1801 7556
SRU2_PIN1	0x1C11	SRU2 Pin Assignment 1	0x004D B699
SRU2_PIN2	0x1C12	SRU2 Pin Assignment 2	0x0045 0000
SRU2_PBEN0	0x1C20	SRU2 Pin Enable 0	0x0D00 C28B

Register Listing

Register Mnemonic	Address	Description	Reset
SRU2_PBEN1	0x1C21	SRU2 Pin Enable 1	0x0021 03CE
SRU2_PBEN2	0x1C22	SRU2 Pin Enable 2	0x0018 5964
DAI Shift Registers			
SRU_CLK_SHREG	0x24F1	SRU Shift Register Clock Control	0x0000 0272
SRU_DAT_SHREG	0x24F2	SRU Shift Register Data Control	0x0000 0012
DAI Interrupt Registers			
DAI_IMASK_FE	0x2480	DAI Falling Edge Interrupt Mask	0x0
DAI_IMASK_RE	0x2481	DAI Rising Edge Interrupt Mask	0x0
DAI_IRPTL_PRI	0x2484	DAI Interrupt Latch Priority	0x0
DAI_IRPTL_H	0x2488	DAI High Priority Interrupt Latch	0x0
DAI_IRPTL_L	0x2489	DAI Low Priority Interrupt Latch	0x0
DAI_IRPTL_HS	0x248C	Shadow DAI High Priority Interrupt Latch	0x0
DAI_IRPTL_LS	0x248D	Shadow DTDMAI Low Priority Interrupt Latch	0x0
DPI Interrupt Registers			
DPI_IRPTL	0x1C32	DPI Interrupt Latch	0x0
DPI_IRPTL_SH	0x1C33	DPI Shadow Priority Latch	0x0
DPI_IMASK_FE	0x1C34	DPI Falling Edge Interrupt Mask	0x0
DPI_IMASK_RE	0x1C35	DPI Rising Edge Interrupt Mask	0x0
DAI/DPI Pin Buffer Status Registers			
DAI_PIN_STAT	0x24B9	DAI Pin Status	0x000F FFFF
DPI_PIN_STAT	0x1C31	DPI Pin Status	0x0000 3FFF
Input Data Port Registers			
IDP_CTL0	0x24B0	IDP Control 0	0x0
IDP_CTL1	0x24B2	IDP Control 1	0x0000 FFFF
IDP_CTL2	0x24B3	IDP Control 2	0x0
IDP_PP_CTL	0x24B1	Parallel Data Acquisition Port Control Register	0x0
IDP_FIFO	0x24D0	IDP FIFO Packing Mode	0x0

Register Mnemonic	Address	Description	Reset
DAI_STAT0	0x24B8	IDP Status	0x0
DAI_STAT1	0x24BA	IDP Status	0x0
Input Data Port DMA Parameter Registers			
IDP_DMA_I0	0x2400	IDP DMA Channel 0 Index	0x0
IDP_DMA_I1	0x2401	IDP DMA Channel 1 Index	0x0
IDP_DMA_I2	0x2402	IDP DMA Channel 2 Index	0x0
IDP_DMA_I3	0x2403	IDP DMA Channel 3 Index	0x0
IDP_DMA_I4	0x2404	IDP DMA Channel 4 Index	0x0
IDP_DMA_I5	0x2405	IDP DMA Channel 5 Index	0x0
IDP_DMA_I6	0x2406	IDP DMA Channel 6 Index	0x0
IDP_DMA_I7	0x2407	IDP DMA Channel 7 Index	0x0
IDP_DMA_I0A	0x2408	IDP DMA Channel 0 Index A for Ping Pong DMA	0x0
IDP_DMA_I1A	0x2409	IDP DMA Channel 1 Index A for Ping Pong DMA	0x0
IDP_DMA_I2A	0x240A	IDP DMA Channel 2 Index A for Ping Pong DMA	0x0
IDP_DMA_I3A	0x240B	IDP DMA Channel 3 Index A for Ping Pong DMA	0x0
IDP_DMA_I4A	0x240C	IDP DMA Channel 4 Index A for Ping Pong DMA	0x0
IDP_DMA_I5A	0x240D	IDP DMA Channel 5 Index A for Ping Pong DMA	0x0
IDP_DMA_I6A	0x240E	IDP DMA Channel 6 Index A for Ping Pong DMA	0x0
IDP_DMA_I7A	0x240F	IDP DMA Channel 7 Index A for Ping Pong DMA	0x0
IDP_DMA_I0B	0x2418	IDP DMA Channel 0 Index B for Ping Pong DMA	0x0
IDP_DMA_I1B	0x2419	IDP DMA Channel 1 Index B for Ping Pong DMA	0x0
IDP_DMA_I2B	0x241A	IDP DMA Channel 2 Index B for Ping Pong DMA	0x0
IDP_DMA_I3B	0x241B	IDP DMA Channel 3 Index B for Ping Pong DMA	0x0
IDP_DMA_I4B	0x241C	IDP DMA Channel 4 Index B for Ping Pong DMA	0x0
IDP_DMA_I5B	0x241D	IDP DMA Channel 5 Index B for Ping Pong DMA	0x0
IDP_DMA_I6B	0x241E	IDP DMA Channel 6 Index B for Ping Pong DMA	0x0
IDP_DMA_I7B	0x241F	IDP DMA Channel 7 Index B for Ping Pong DMA	0x0

Register Listing

Register Mnemonic	Address	Description	Reset
IDP_DMA_M0	0x2410	IDP DMA Channel 0 Modify	0x0
IDP_DMA_M1	0x2411	IDP DMA Channel 1 Modify	0x0
IDP_DMA_M2	0x2412	IDP DMA Channel 2 Modify	0x0
IDP_DMA_M3	0x2413	IDP DMA Channel 3 Modify	0x0
IDP_DMA_M4	0x2414	IDP DMA Channel 4 Modify	0x0
IDP_DMA_M5	0x2415	IDP DMA Channel 5 Modify	0x0
IDP_DMA_M6	0x2416	IDP DMA Channel 6 Modify	0x0
IDP_DMA_M7	0x2417	IDP DMA Channel 7 Modify	0x0
IDP_DMA_C0	0x2420	IDP DMA Channel 0 Count	0x0
IDP_DMA_C1	0x2421	IDP DMA Channel 1 Count	0x0
IDP_DMA_C2	0x2422	IDP DMA Channel 2 Count	0x0
IDP_DMA_C3	0x2423	IDP DMA Channel 3 Count	0x0
IDP_DMA_C4	0x2424	IDP DMA Channel 4 Count	0x0
IDP_DMA_C5	0x2425	IDP DMA Channel 5 Count	0x0
IDP_DMA_C6	0x2426	IDP DMA Channel 6 Count	0x0
IDP_DMA_C7	0x2427	IDP DMA Channel 7 Count	0x0
IDP_DMA_PC0	0x2428	IDP DMA Channel 0 Ping Pong Count	0x0
IDP_DMA_PC1	0x2429	IDP DMA Channel 1 Ping Pong Count	0x0
IDP_DMA_PC2	0x242A	IDP DMA Channel 2 Ping Pong Count	0x0
IDP_DMA_PC3	0x242B	IDP DMA Channel 3 Ping Pong Count	0x0
IDP_DMA_PC4	0x242C	IDP DMA Channel 4 Ping Pong Count	0x0
IDP_DMA_PC5	0x242D	IDP DMA Channel 5 Ping Pong Count	0x0
IDP_DMA_PC6	0x242E	IDP DMA Channel 6 Ping Pong Count	0x0
IDP_DMA_PC7	0x242F	IDP DMA Channel 7 Ping Pong Count	0x0
Precision Clock Generator Registers			
PCG_CTLA0	0x24C0	Precision Clock A Control 0	0x0
PCG_CTLA1	0x24C1	Precision Clock A Control 1	0x0

Register Mnemonic	Address	Description	Reset
PCG_CTLB0	0x24C2	Precision Clock B Control 0	0x0
PCG_CTLB1	0x24C3	Precision Clock B Control 1	0x0
PCG_CTLC0	0x24C6	Precision Clock C Control 0	0x0
PCG_CTLC1	0x24C7	Precision Clock C Control 1	0x0
PCG_CTLD0	0x24C8	Precision Clock D Control 0	0x0
PCG_CTLD1	0x24C9	Precision Clock D Control 1	0x0
PCG_PW1	0x24C4	Precision Clock Pulse Width Control 1	0x0
PCG_PW2	0x24CA	Precision Clock Pulse Width Control 2	0x0
PCG_SYNC1	0x24C5	Precision Clock Frame Sync Synchronization 1	0x0
PCG_SYNC2	0x24CB	Precision Clock Frame Sync Synchronization 2	0x0
Programmable Interrupt Priority Control Registers			
PICR0	0x2200	Programmable Interrupt Priority Control 0	0x0A41 8820
PICR1	0x2201	Programmable Interrupt Priority Control 1	0x16A4 A0E6
PICR2	0x2202	Programmable Interrupt Priority Control 2	0x2307 B9AC
PICR3	0x2203	Programmable Interrupt Priority Control 3	0x0000 0012
Pulse Width Modulation Registers			
PWMGCTL	0x3800	PWM Global Control	0x0
PWMGSTAT	0x3801	PWM Global Status	0x0
PWMCTL0	0x3000	PWM Control 0	0x0
PWMSTAT0	0x3001	PWM Status 0	0x0009
PWMPERIOD0	0x3002	PWM Period 0	0x0
PWMDT0	0x3003	PWM Dead Time 0	0x0
PWMA0	0x3005	PWM Channel A Duty Control 0	0x0
PWMB0	0x3006	PWM Channel B Duty Control 0	0x0
PWMSEG0	0x3008	PWM Output Enable 0	0x0
PWMAL0	0x300A	PWM Channel AL Duty Control 0	0x0
PWMBL0	0x300B	PWM Channel BL Duty Control 0	0x0

Register Listing

Register Mnemonic	Address	Description	Reset
PWMDBG0	0x300E	PWM Debug Status 0	0x0
PWMPOLO	0x300F	PWM Output Polarity Select 0	0x00FF
PWMCTL1	0x3010	PWM Control 1	0x0
PWMSTAT1	0x3011	PWM Status 1	0x0009
PWMPERIOD1	0x3012	PWM Period 1	0x0
PWMDT1	0x3013	PWM Dead Time 1	0x0
PWMA1	0x3015	PWM Channel A Duty Control 1	0x0
PWMB1	0x3016	PWM Channel B Duty Control 1	0x0
PWMSEG1	0x3018	PWM Output Enable 1	0x0
PWMAL1	0x301A	PWM Channel AL Duty Control 1	0x0
PWMBL1	0x301B	PWM Channel BL Duty Control 1	0x0
PWMDBG1	0x301E	PWM Debug Status 1	0x0
PWMPOL1	0x301F	PWM Output Polarity Select 1	0x00FF
PWMCTL2	0x3400	PWM Control 2	0x0
PWMSTAT2	0x3401	PWM Status 2	0x0009
PWMPERIOD2	0x3402	PWM Period 2	0x0
PWMDT2	0x3403	PWM Dead Time 2	0x0
PWMA2	0x3405	PWM Channel A Duty Control 2	0x0
PWMB2	0x3406	PWM Channel B Duty Control 2	0x0
PWMSEG2	0x3408	PWM Output Enable 2	0x0
PWMAL2	0x340A	PWM Channel AL Duty Control 2	0x0
PWMBL2	0x340B	PWM Channel BL Duty Control 2	0x0
PWMDBG2	0x340E	PWM Debug Status 2	0x0
PWMPOL2	0x340F	PWM Output Polarity Select 2	0x00FF
PWMCTL3	0x3410	PWM Control 3	0x0
PWMSTAT3	0x3411	PWM Status 3	0x0009
PWMPERIOD3	0x3412	PWM Period 3	0x0

Register Mnemonic	Address	Description	Reset
PWMDT3	0x3413	PWM Dead Time 3	0x0
PWMA3	0x3415	PWM Channel A Duty Control 3	0x0
PWMB3	0x3416	PWM Channel B Duty Control 3	0x0
PWMSEG3	0x3418	PWM Output Enable 3	0x0
PWMAL3	0x341A	PWM Channel AL Duty Control 3	0x0
PWMBL3	0x341B	PWM Channel BL Duty Control 3	0x0
PWMDBG3	0x341E	PWM Debug Status 3	0x0
PWMPOL3	0x341F	PWM Output Polarity Select 3	0x00FF
Memory-to-Memory DMA Registers			
MTMCTL	0x2C01	Memory-to-Memory DMA Control	0x0
IIMTMW	0x2C10	MTM DMA Destination Index	0x0
IIMTMR	0x2C11	MTM DMA Source Index	0x0
IMMTMW	0x2C0E	MTM DMA Destination Modify	0x0
IMMTMR	0x2C0F	MTM DMA Source Modify	0x0
CMTMW	0x2C16	MTM DMA Destination Count	0x0
CMTMR	0x2C17	MTM DMA Source Count	0x0
FFT Accelerator Registers			
FFTCTL1	0x5300	FFT Global Control	0x0
FFTCTL2	0x530C	Channel Control	0x0
FTMACSTAT	0x5302	MAC Status	0x0
FTDADDR	0x5303	Debug Address	0x0
FTDDATA	0x5304	Debug Data	0x0
IIFT	0x5310	Input Index	0x0
IMFFT	0x5311	Input Modifier	0x0
ICFFT	0x5312	Input Count	0x0
ILFFT	0x5313	Input Length	0x0
IBFFT	0x5314	Input Base	0x0

Register Listing

Register Mnemonic	Address	Description	Reset
ICPFFT	0x5315	Input Chain Pointer	0x0
OIFFT	0x5318	Output Index	0x0
OMFFT	0x5319	Output Modifier	0x0
OCFFT	0x531A	Output Count	0x0
OLFFT	0x531B	Output Length	0x0
OBFFT	0x531C	Output Base	0x0
OCPFFT	0x531D	Output Chain Pointer	0x0
FFTDMASTAT	0x5320	DMA Status	0x0
FTSHDMASTAT	0x5321	DMA Shadow Status	0x0
FIR Accelerator Registers			
FIRCTL1	0x5000	FIR Global Control	0x0
FIRDMASTAT	0x5001	DMA Status Reg	0x0
FIRMACSTAT	0x5002	MAC Status	0x0
FIRDEBUGCTL	0x5004	Debug Control	0x0
FIRDBGADDR	0x5005	Debug Address	0x0
FIRDBGWRDATA	0x5006	Debug Data Write	0x0
FIRDBGRDDATA	0x5007	Debug Data Read	0x0
FIRCTL2	0x5010	Channel Control	0x0
IIFIR	0x5011	Input Data Index	0x0
IMFIR	0x5012	Input Data Modifier	0x0
ICFIR	0x5013	Input Data Count	0x0
IBFIR	0x5014	Input Data Base	0x0
OIFIR	0x5015	Output Data Index	0x0
OMFIR	0x5016	Output Data Modifier	0x0
OCFIR	0x5017	Output Data Count	0x0
OBFIR	0x5018	Output Data Base	0x0
CIFIR	0x5019	Coefficient Index	0x0

Register Mnemonic	Address	Description	Reset
CMFIR	0x501A	Coefficient Modifier	0x0
CCFIR	0x501B	Coefficient Count	0x0
CPFIR	0x501C	Chain Pointer	0x0
IIR Accelerator Registers			
IIRCTL1	0x5200	IIR Global Control	0x0
IIRDMASTAT	0x5201	DMA Status Reg	0x0
IIRMACSTAT	0x5202	MAC Status	0x0
IIRDEBUGCTL	0x5203	Debug Control	0x0
IIRDBGADDR	0x5204	Debug Address	0x0
IIRDBGWRDATA_L	0x5205	Debug Data Write LS 32 Bits	0x0
IIRDBGWRDATA_H	0x5206	Debug Data Write MS 8 Bits	0x0
IIRDBGRDDATA_L	0x5207	Debug Data Read LS 32 Bits	0x0
IIRDBGRDDATA_H	0x5208	Debug Data Read MS 8 Bits	0x0
IIRCTL2	0x5210	Channel Control	0x0
IIIIR	0x5211	Input Data Index	0x0
IMIIR	0x5212	Input Data Modifier	0x0
ICIIIR	0x5213	Input Data Count	0x0
IBIIR	0x5214	Input Data Base	0x0
OIIIR	0x5215	Output Data Index	0x0
OMIIR	0x5216	Output Data Modifier	0x0
OCIIR	0x5217	Output Data Count	0x0
OBIIIR	0x5218	Output Data Base	0x0
CIIIR	0x5219	Coefficient Index	0x0
CMIIR	0x521A	Coefficient Modifier	0x0
CCIIR	0x521B	Coefficient Count	0x0
CPIIR	0x521C	Chain Pointer	0x0
S/PDIF Transmit Registers			

Register Listing

Register Mnemonic	Address	Description	Reset
DITCTL	0x24A0	Digital Interface Transmit Control	0x0
S/PDIF Channel Status Registers			
DITCHANA0	0x24A1	Transmit CH0 Subframe A	0x0
DITCHANA1	0x24D4	Transmit CH1 Subframe A	0x0
DITCHANA2	0x24D5	Transmit CH2 Subframe A	0x0
DITCHANA3	0x24D6	Transmit CH3 Subframe A	0x0
DITCHANA4	0x24D7	Transmit CH4 Subframe A	0x0
DITCHANA5	0x24D8	Transmit CH5 Subframe A	0x0
DITCHANB0	0x24A2	Transmit CH0 Subframe B	0x0
DITCHANB1	0x24DA	Transmit CH1 Subframe B	0x0
DITCHANB2	0x24DB	Transmit CH2 Subframe B	0x0
DITCHANB3	0x24DC	Transmit CH3 Subframe B	0x0
DITCHANB4	0x24DD	Transmit CH4 Subframe B	0x0
DITCHANB5	0x24DE	Transmit CH5 Subframe B	0x0
S/PDIF User Bit Status Registers			
DITUSRBITA0	0x24E0	Transmit User Bit CH0 Subframe A	0x0
DITUSRBITA1	0x24E1	Transmit User Bit CH1 Subframe A	0x0
DITUSRBITA2	0x24E2	Transmit User Bit CH2 Subframe A	0x0
DITUSRBITA3	0x24E3	Transmit User Bit CH3 Subframe A	0x0
DITUSRBITA4	0x24E4	Transmit User Bit CH4 Subframe A	0x0
DITUSRBITA5	0x24E5	Transmit User Bit CH5 Subframe A	0x0
DITUSRBITB0	0x24E8	Transmit User Bit CH0 Subframe B	0x0
DITUSRBITB1	0x24E9	Transmit User Bit CH1 Subframe B	0x0
DITUSRBITB2	0x24EA	Transmit User Bit CH2 Subframe B	0x0
DITUSRBITB3	0x24EB	Transmit User Bit CH3 Subframe B	0x0
DITUSRBITB4	0x24EC	Transmit User Bit CH4 Subframe B	0x0

Register Mnemonic	Address	Description	Reset
DITUSRBITB5	0x24ED	Transmit User Bit CH5 Subframe B	0x0
DITUSRUPD	0x24EF	Transmit User Bit Update	0x0
S/PDIF Receiver Registers			
DIRCTL	0x24A8	Receiver Control	0x0
DIRSTAT	0x24A9	Receiver Status	0x20
DIRCHANL	0x24AA	Receiver Left Channel Status	0x0
DIRCHANR	0x24AB	Receiver Right Channel Status	0x0
Sample Rate Converter Registers			
SRCCCTL0	0x2490	SRC0 Control	0x0
SRCCCTL1	0x2491	SRC1 Control	0x0
SRCMUTE	0x2492	SRC Mute	0x0
SRCRAT0	0x2498	SRC0 Output to Input Ratio	0x8000 8000
SRCRAT1	0x2499	SRC1 Output to Input Ratio	0x8000 8000
UART Registers			
UART0LCR	0x3C03	UART0 Line Control	0x0
UART0LSR	0x3C05	UART0 Line Status	0x60
UART0THR	0x3C00	UART0 Transmit Hold	0x0
UART0RBR	0x3C00	UART0 Receive Buffer	0x0
UART0IER	0x3C01	UART0 Interrupt Enable	0x0
UART0IIR	0x3C02	UART0 Interrupt ID	0x1
UART0DLL	0x3C00	UART0 Divisor Latch Low	0x0
UART0DLH	0x3C01	UART0 Divisor Latch High	0x0
UART0SCR	0x3C07	UART0 Scratch	Undefined
UART0MODE	0x3C04	UART0 Mode	0x20
RXI_UAC0	0x3E00	UART Receive Index Register	0x0
RXM_UAC0	0x3E01	UART Receive Modifier Register	0x0
RXC_UAC0	0x3E02	UART Receive Count Register	0x0

Register Listing

Register Mnemonic	Address	Description	Reset
RXCP_UAC0	0x3E03	UART Receive Chain Pointer Register	0x0
TXI_UAC0	0x3F00	UART Transmit Index Register	0x0
TXM_UAC0	0x3F01	UART Transmit Modifier Register	0x0
TXC_UAC0	0x3F02	UART Receive Count Register	0x0
TXCP_UAC0	0x3F03	UART Transmit Chain Pointer Register	0x0
UART0TXCTL	0x3F04	UART0 DMA TX Control	0x0
UART0RXCTL	0x3E04	UART0 DMA RX Control	0x0
UART0TXSTAT	0x3F05	UART0 DMA TX Status	0x0
UART0RXSTAT	0x3E05	UART0 DMA RX Status	0x0
TWI Registers			
TWIDIV	0x4400	SCL Clock Divider	0x0
TWIMITR	0x4404	Master Internal Time Reference	0x0
TWISCTL	0x4408	Slave Mode Control	0x0
TWISSTAT	0x440C	Slave Mode Status	0x0
TWISADDR	0x4410	Slave Mode Address	0x0
TWIMCTL	0x4414	Master Mode Control	0x0
TWIMSTAT	0x4418	Master Mode Status	0x0
TWIMADDR	0x441C	Master Mode Address	0x0
TWIIRPTL	0x4420	Interrupt Latch	0x0
TWIIMASK	0x4424	Interrupt Mask	0x0
TWIFIFOCTL	0x4428	FIFO Control	0x0
TWIFIFOSTAT	0x442C	FIFO Status	0x0
TXTWI 8	0x4480	8-bit FIFO Transmit	0x0
TXTWI 16	0x4484	16-bit FIFO Transmit	0x0
RXTWI8	0x4488	8-Bit FIFO Receive	0x0
RXTWI16	0x448C	16-Bit FIFO Receive	0x0

Register Mnemonic	Address	Description	Reset
Link Port Registers			
LCTL0	0x4C00	Link Port 0 Control	0x0
LSTST0	0x4C01	Link Port 0 Status	0x0
TXLB0	0x4C08	Link Port 0 TX Buffer	Undefined
RXLB0	0x4C09	Link Port 0 RX Buffer	Undefined
TXLB0_IN_SHADOW	0x4C0D	Link Port 0 Shadow Input TX Buffer	0x0
TXLB0_OUT_SHADOW	0x4C0C	Link Port 0 Shadow Output TX Buffer	0x0
RXLB0_IN_SHADOW	0x4C0B	Link Port 0 Shadow Input RX Buffer	0x0
RXLB0_OUT_SHADOW	0x4C0A	Link Port 0 Shadow Output RX Buffer	0x0
LSTAT0_SHADOW	0x4C03	Link Port 0 Shadow Status	0x0
IILB0	0x4C18	Link Port 0 Internal Index	0x0
IMLB0	0x4C19	Link Port 0 Internal Modifier	0x0
CLB0	0x4C1A	Link Port 0 Internal Count	0x0
CPLB0	0x4C1B	Link Port 0 Chain Pointer	0x0
LCTL1	0x4C20	Link Port 1 Control	0x0
LSTST1	0x4C21	Link Port 1 Status	0x0
TXLB1	0x4C28	Link Port 1 TX Buffer	Undefined
RXLB1	0x4C29	Link Port 1 RX Buffer	Undefined
TXLB1_IN_SHADOW	0x4C2D	Link Port 1 Shadow Input TX Buffer	0x0
TXLB1_OUT_SHADOW	0x4C2C	Link Port 1 Shadow Output TX Buffer	0x0
RXLB1_IN_SHADOW	0x4C2B	Link Port 1 Shadow Input RX Buffer	0x0

Register Listing

Register Mnemonic	Address	Description	Reset
RXLB1_OUT_SHADOW	0x4C2A	Link Port 1 Shadow Output RX Buffer	0x0
LSTAT1_SHADOW	0x4C23	Link Port 1 Shadow Status	0x0
IILB1	0x4C38	Link Port 1 Internal Index	0x0
IMLB1	0x4C39	Link Port 1 Internal Modifier	0x0
CLB1	0x4C3A	Link Port 1 Internal Count	0x0
CPLB1	0x4C3B	Link Port 1 Chain Pointer	0x0
MLB Registers			
MLB_DCCR	0x4100	MLB Device Control	0x0
MLB_SSCR	0x4101	MLB System Status Configuration	0x0
MLB_SDCR	0x4102	System Data Configuration	0x0
MLB_SMCR	0x4103	System Interrupt Mask	0x00000060
MLB_VCCR	0x4107	Version Control Config (Contains the IP Version)	0x202
MLB_SBCR	0x4108	Synchronous Base Address	0x0
MLB_ABCR	0x4109	Asynchronous Base Address	0x0
MLB_CBCR	0x410A	Control Base Address	0x0
MLB_IBCR	0x410B	Isochronous Base Address	0x0
MLB_CICR	0x410C	Channel Interrupt Status	0x0
MLB_CECR0	0x4110	Channel 0 Control	0x0
MLB_CSCR0	0x4111	Channel 0 Status	0x8000 0000
MLB_CCBCR0	0x4112	Channel 0 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR0	0x4113	Channel 0 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR0	0x41A0	Channel 0 Local Channel Buffer Control	0x0040 0000
MLB_CECR1	0x4114	Channel 1 Control	0x0
MLB_CSCR1	0x4115	Channel 1 Status	0x8000 0000
MLB_CCBCR1	0x4116	Channel 1 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR1	0x4117	Channel 1 Next Buffer (TX Buffer in I/O Mode)	0x0

Register Mnemonic	Address	Description	Reset
MLB_LCBR1	0x41A1	Channel 1 Local Channel Buffer Control	0x0040 0001
MLB_CECR2	0x4118	Channel 2 Control	0x0
MLB_CSCR2	0x4119	Channel 2 Status	0x8000 0000
MLB_CCBCR2	0x411A	Channel 2 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR2	0x411B	Channel 2 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBR2	0x41A2	Channel 2 Local Channel Buffer Control	0x0040 0002
MLB_CECR3	0x411C	Channel 3 Control	0x0
MLB_CSCR3	0x411D	Channel 3 Status	0x8000 0000
MLB_CCBCR3	0x411E	Channel 3 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR3	0x411F	Channel 3 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBR3	0x41A3	Channel 3 Local Channel Buffer Control	0x0040 0003
MLB_CECR4	0x4120	Channel 4 Control	0x0
MLB_CSCR4	0x4121	Channel 4 Status	0x8000 0000
MLB_CCBCR4	0x4122	Channel 4 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR4	0x4123	Channel 4 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBR4	0x41A4	Channel 4 Local Channel Buffer Control	0x0040 0004
MLB_CECR5	0x4124	Channel 5 Control	0x0
MLB_CSCR5	0x4125	Channel 5 Status	0x8000 0000
MLB_CCBCR5	0x4126	Channel 5 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR5	0x4127	Channel 5 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBR5	0x41A5	Channel 5 Local Channel Buffer Control	0x0040 0005
MLB_CECR6	0x4128	Channel 6 Control	0x0
MLB_CSCR6	0x4129	Channel 6 Status	0x8000 0000
MLB_CCBCR6	0x412A	Channel 6 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR6	0x412B	Channel 6 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBR6	0x41A6	Channel 6 Local Channel Buffer Control	0x0040 0006
MLB_CECR7	0x412C	Channel 7 Control	0x0

Register Listing

Register Mnemonic	Address	Description	Reset
MLB_CSCR7	0x412D	Channel 7 Status	0x8000 0000
MLB_CCBCR7	0x412E	Channel 7 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR7	0x412F	Channel 7 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR7	0x41A7	Channel 7 Local Channel Buffer Control	0x0040 0007
MLB_CECR8	0x4130	Channel 8 Control	0x0
MLB_CSCR8	0x4131	Channel 8 Status	0x8000 0000
MLB_CCBCR8	0x4132	Channel 8 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR8	0x4133	Channel 8 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR8	0x41A8	Channel 8 Local Channel Buffer Control	0x0040 0008
MLB_CECR9	0x4134	Channel 9 Control	0x0
MLB_CSCR9	0x4135	Channel 9 Status	0x8000 0000
MLB_CCBCR9	0x4136	Channel 9 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR9	0x4137	Channel 9 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR9	0x41A9	Channel 9 Local Channel Buffer Control	0x0040 0009
MLB_CECR10	0x4138	Channel 10 Control	0x0
MLB_CSCR10	0x4139	Channel 10 Status	0x8000 0000
MLB_CCBCR10	0x413A	Channel 10 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR10	0x413B	Channel 10 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR10	0x41AA	Channel 10 Local Channel Buffer Control	0x0040 000A
MLB_CECR11	0x413C	Channel 11 Control	0x0
MLB_CSCR11	0x413D	Channel 11 Status	0x8000 0000
MLB_CCBCR11	0x413E	Channel 11 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR11	0x413F	Channel 11 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR11	0x41AB	Channel 11 Local Channel Buffer Control	0x0040 000B
MLB_CECR12	0x4140	Channel 12 Control	0x0
MLB_CSCR12	0x4141	Channel 12 Status	0x8000 0000
MLB_CCBCR12	0x4142	Channel 12 Current Buffer (RX Buffer in I/O Mode)	0x0

Register Mnemonic	Address	Description	Reset
MLB_CNBCR12	0x4143	Channel 12 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR12	0x41AC	Channel 12 Local Channel Buffer Control	0x0040 000C
MLB_CECR13	0x4144	Channel 13 Control	0x0
MLB_CSCR13	0x4145	Channel 13 Status	0x8000 0000
MLB_CCBCR13	0x4146	Channel 13 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR13	0x4147	Channel 13 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR13	0x41AD	Channel 13 Local Channel Buffer Control	0x0040 000D
MLB_CECR14	0x4148	Channel 14 Control	0x0
MLB_CSCR14	0x4149	Channel 14 Status	0x8000 0000
MLB_CCBCR14	0x414A	Channel 14 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR14	0x414B	Channel 14 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR14	0x41AE	Channel 14 Local Channel Buffer Control	0x0040 000E
MLB_CECR15	0x414C	Channel 15 Control	0x0
MLB_CSCR15	0x414D	Channel 15 Status	0x8000 0000
MLB_CCBCR15	0x414E	Channel 15 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR15	0x414F	Channel 15 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR15	0x41AF	Channel 15 Local Channel Buffer Control	0x0040 000F
MLB_CECR16	0x4150	Channel 16 Control	0x0
MLB_CSCR16	0x4151	Channel 16 Status	0x8000 0000
MLB_CCBCR16	0x4152	Channel 16 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR16	0x4153	Channel 16 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR16	0x41B0	Channel 16 Local Channel Buffer Control	0x0040 0010
MLB_CECR17	0x4154	Channel 17 Control	0x0
MLB_CSCR17	0x4155	Channel 17 Status	0x8000 0000
MLB_CCBCR17	0x4156	Channel 17 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR17	0x4157	Channel 17 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR17	0x41B1	Channel 17 Local Channel Buffer Control	0x0040 0011

Register Listing

Register Mnemonic	Address	Description	Reset
MLB_CECR18	0x4158	Channel 18 Control	0x0
MLB_CSCR18	0x4159	Channel 18 Status	0x8000 0000
MLB_CCBCR18	0x415A	Channel 18 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR18	0x415B	Channel 18 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR18	0x41B2	Channel 18 Local Channel Buffer Control	0x0040 0012
MLB_CECR19	0x415C	Channel 19 Control	0x0
MLB_CSCR19	0x415D	Channel 19 Status	0x8000 0000
MLB_CCBCR19	0x415E	Channel 19 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR19	0x415F	Channel 19 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR19	0x41B3	Channel 19 Local Channel Buffer Control	0x0040 0013
MLB_CECR20	0x4160	Channel 20 Control	0x0
MLB_CSCR20	0x4161	Channel 20 Status	0x8000 0000
MLB_CCBCR20	0x4162	Channel 20 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR20	0x4163	Channel 20 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR20	0x41B4	Channel 20 Local Channel Buffer Control	0x0040 0014
MLB_CECR21	0x4164	Channel 21 Control	0x0
MLB_CSCR21	0x4165	Channel 21 Status	0x8000 0000
MLB_CCBCR21	0x4166	Channel 21 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR21	0x4167	Channel 21 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR21	0x41B5	Channel 21 Local Channel Buffer Control	0x0040 0015
MLB_CECR22	0x4168	Channel 22 Control	0x0
MLB_CSCR22	0x4169	Channel 22 Status	0x8000 0000
MLB_CCBCR22	0x416A	Channel 22 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR22	0x416B	Channel 22 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR22	0x41B6	Channel 22 Local Channel Buffer Control	0x0040 0016
MLB_CECR23	0x416C	Channel 23 Control	0x0
MLB_CSCR23	0x416D	Channel 23 Status	0x8000 0000

Register Mnemonic	Address	Description	Reset
MLB_CCBCR23	0x416E	Channel 23 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR23	0x416F	Channel 23 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR23	0x41B7	Channel 23 Local Channel Buffer Control	0x0040 0017
MLB_CECR24	0x4170	Channel 24 Control	0x0
MLB_CSCR24	0x4171	Channel 24 Status	0x8000 0000
MLB_CCBCR24	0x4172	Channel 24 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR24	0x4173	Channel 24 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR24	0x41B8	Channel 24 Local Channel Buffer Control	0x0040 0018
MLB_CECR25	0x4174	Channel 25 Control	0x0
MLB_CSCR25	0x4175	Channel 25 Status	0x8000 0000
MLB_CCBCR25	0x4176	Channel 25 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR25	0x4177	Channel 25 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR25	0x41B9	Channel 25 Local Channel Buffer Control	0x0040 0019
MLB_CECR26	0x4178	Channel 26 Control	0x0
MLB_CSCR26	0x4179	Channel 26 Status	0x8000 0000
MLB_CCBCR26	0x417A	Channel 26 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR26	0x417B	Channel 26 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR26	0x41BA	Channel 26 Local Channel Buffer Control	0x0040 001A
MLB_CECR27	0x417C	Channel 27 Control	0x0
MLB_CSCR27	0x417D	Channel 27 Status	0x8000 0000
MLB_CCBCR27	0x417E	Channel 27 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR27	0x417F	Channel 27 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBCR27	0x41BB	Channel 27 Local Channel Buffer Control	0x0040 001B
MLB_CECR28	0x4180	Channel 28 Control	0x0
MLB_CSCR28	0x4181	Channel 28 Status	0x8000 0000
MLB_CCBCR28	0x4182	Channel 28 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR28	0x4183	Channel 28 Next Buffer (TX Buffer in I/O Mode)	0x0

Register Listing

Register Mnemonic	Address	Description	Reset
MLB_LCBR28	0x41BC	Channel 28 Local Channel Buffer Control	0x0040 001C
MLB_CECR29	0x4184	Channel 29 Control	0x0
MLB_CSCR29	0x4185	Channel 29 Status	0x8000 0000
MLB_CCBCR29	0x4186	Channel 29 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR29	0x4187	Channel 29 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBR29	0x41BD	Channel 29 Local Channel Buffer Control	0x0040 001D
MLB_CECR30	0x4188	Channel 30 Control	0x0
MLB_CSCR30	0x4189	Channel 30 Status	0x8000 0000
MLB_CCBCR30	0x418A	Channel 30 Current Buffer (RX Buffer in I/O Mode)	0x0
MLB_CNBCR30	0x418B	Channel 30 Next Buffer (TX Buffer in I/O Mode)	0x0
MLB_LCBR30	0x41BE	Channel 30 Local Channel Buffer Control	0x0040 001E
Shift Register Register			
SR_CTL	0x24F0	Shift Register Control	0x0
Watchdog Timer Registers			
WDTCTL	0x5400	Watchdog Timer Control	0x0
WDTCURCNT	0x5401	Watchdog Timer Current Count	0x0
WDTTRIP	0x5402	Watchdog Timer Trip	0x0
WDTCNT	0x5403	Watchdog Timer Count	0x0
WDTSTATUS	0x5404	Watchdog Timer Status	0x0
WDTUNLOCK	0x5405	Watchdog Timer Unlock	0x0
WDTCLKSEL	0x5408	Watchdog Timer Clock Select	0x0
Real-Time Clock Registers			
RTC_CLOCK	0x4CA0	Real-Time Clock Clock	Undefined
RTC_ALARM	0x4CA1	Real-Time Clock Alarm	Undefined
RTC_CTL	0x4CA2	Real-Time Clock Control	0x0
RTC_STAT	0x4CA3	Real-Time Clock Status	0x0
RTC_STPWTCH	0x4CA4	Real-Time Clock Stopwatch	Undefined

Registers Reference

Register Mnemonic	Address	Description	Reset
RTC_INIT	0x4CA6	Real-Time Clock Initialization	0x0
RTC_INITSTAT	0x4CA7	Real-Time Clock Initialization Status	0x0

Register Listing

B PERIPHERAL INTERRUPT CONTROL

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

Interrupt Latency

The following

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

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

Interrupt Acknowledge



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

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

```
ISR_SPI_Routine:  
R0 = dm(i0,m0);  
dm(TXSPI) = R0; /* write to SPI data buffer */  
R0 = dm(SPICTL); /* force dummy read to terminate write */  
rti;
```

```
ISR_PWM_Routine:  
r1=PWM_STAT3;  
dm(PWMGSTAT)=r1; /* W1C to PWM status reg */  
r0=dm(PWMGSTAT); /* force dummy read to terminate write */  
rti;
```

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

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

Interrupt Acknowledge

As previously discussed in this manual, interrupt driven I/O is advantageous in that programs do not need to poll the core. When an interrupt is triggered, the sequencer typically finishes the current instruction and jump to the IVT (interrupt vector table). From IVT the address then typically vectors to the ISR routine. The sequencer jumps into this routine,

performs program execution and then exits the routine by executing the RTI (return from interrupt) instruction. However this rule does not apply for all cases as is discussed below.

There are three interrupt acknowledge mechanisms used in an ISR routine:

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

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



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

Interrupt Completion

On SHARC processors, interrupts are generated after *internal transfer completion* (when the DMA count register has expired). However, in some cases the transfer may not have terminated (due to different channel priorities) and valid data still resides in the peripheral's buffer, waiting to be transmitted. To overcome this problem, the interrupt *access completion* mode is introduced. In this mode the interrupt is generated when the last data has left the buffer. [Table B-1](#) provides an overview, for details refer to the specific peripheral's chapter.

Interrupt Priority

Table B-1. Interrupt Acknowledge Mechanisms

Peripheral	Interrupt Source	Interrupt Access Completion	Interrupt Acknowledge
EPDMA1–0	DMA	Yes (ADSP-2137x only)	ISR requires RTI only
MTM	DMA	No	
SPORTs	Core/DMA	Yes	
SPI/SPIB	Core/DMA	Yes	
DAI	IDP, SPDIF, ASRC, MISCA	No	ISR requires read-only-to-clear (ROC-bits) and RTI
DPI	MISCB	No	
UART0	Core/DMA	No	
RTC	Core	No	ROC and RTI
PWM TWI Timer1–0	Core	No	ISR requires write to clear (W1C) and RTI
MLB	Core/DMA	No	W1C and RTI

Interrupt Priority

The following sections provide descriptions of the programmable interrupts that are used in the processors. These registers allow programs to substitute the default interrupts for some other interrupt source.

[Table B-2](#) lists the locations to be programmed in the IOP programmable interrupt control registers (`PICR`) to route an IOP interrupt source to a corresponding processor interrupt location. Note that the shaded area denotes default routing.

[Table B-2](#) also defines the `PICR` bits which should be programmed to select the source for each priority interrupt. Priority programming can be accomplished by changing the sources for each priority interrupt. For

example, if peripheral x should be given high priority, the high priority priority interrupt source should be set as that peripheral (x).

Interrupt Priority

Table B-2. Peripheral Interrupt Controller Routing Table

Interrupt Name	Vector Address	Control Register (PICR)	Default Select Value	Default Function	Priority
POI	0x2C	PICR0[4–0]	0x00	DAIHI interrupt	HIGHEST
P1I ¹	0x30	PICR0[9–5]	0x01	SPI interrupt	
P2I	0x34	PICR0[14–10]	0x02	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 ¹	0x50	PICR1[19–15]	0x09	EPDMA0 interrupt	
P10I	0x54	PICR1[24–20]	0x0A	GP Timer-1 interrupt	
P11I	0x58	PICR1[29–25]	0x0B	SPORT7 interrupt	
P12I	0x5C	PICR2[4–0]	0x0C	DAILI interrupt	
P13I	0x60	PICR2[9–5]	0x0D	EPDMA1 interrupt	
P14I	0x64	PICR2[14–10]	0x0E	DPI interrupt	
P15I	0x68	PICR2[19–15]	0x0F	MTM, DTCP interrupt	
P16I	0x6C	PICR2[24–20]	0x10	SPORT6 interrupt	
P17I	0x70	PICR2[29–25]	0x11	GP timer-2 interrupt	
P18I	0x74	PICR3[4–0]	0x12	SPIB interrupt	LOWEST

Table B-2. Peripheral Interrupt Controller Routing Table (Cont'd)

Interrupt Name	Vector Address	Control Register (PICR)	Default Select Value	Default Function	Priority
UART0RxI	All of these interrupts are not connected by default.	0x13	UART0 receive interrupt		
UART0TxI			0x15	UART0 transmit interrupt	
TWII			0x17	Two wire interface interrupt	
PWMI			0x18	PWM interrupt	
LP0I/RTCI			0x19	Link port 0/RTC interrupt	
LP1I			0x1A	Link port 1 interrupt	
ACC0I			0x1B	Accelerator 01 interrupt	
ACC1I			0x1C	Accelerator 1 interrupt	
MLBI			0x1D	Media local bus interrupt	
Reserved			0x1E	Reserved	
Logic High ²			0x1F	Software option to set IOP interrupts	

- 1 These interrupts have an option to be unmasked at reset. Therefore, the peripherals that boot the processor should be allocated these interrupts: (P1I, P9I).
- 2 Logic high can trigger a software based interrupt on any specific DMA channel (similar to core SFTxI interrupts).

Peripherals with Multiple Interrupt Vector Addresses

The TWI and the UART peripherals have multiple interrupt vector addresses. Both peripherals are already connected to the P14I (DPI) at default. However both have another sets of interrupt latches which are not routed by default. This gives more flexibility to change priority across the programmable interrupts since the DPI does not support high or low priority (unlike DAI).

Interrupt Priority

Priority Interrupt Control Registers (PICRx)

This 32-bit read/write registers, shown in [Figure B-1](#) through [Figure B-4](#), control programmable priority interrupts and default interrupt sources. An example is shown below.

Route PWMI to P17I

```
#include <def21469.h>
ustat1=dm(PICR2);
bit set ustat1 P17I4|P17I3;      /*PWMI 0x18, I4-0= 11000*/
bit clr ustat1 P17I2|P17I1|P17I0;
dm(PICR2)=ustat1;
```

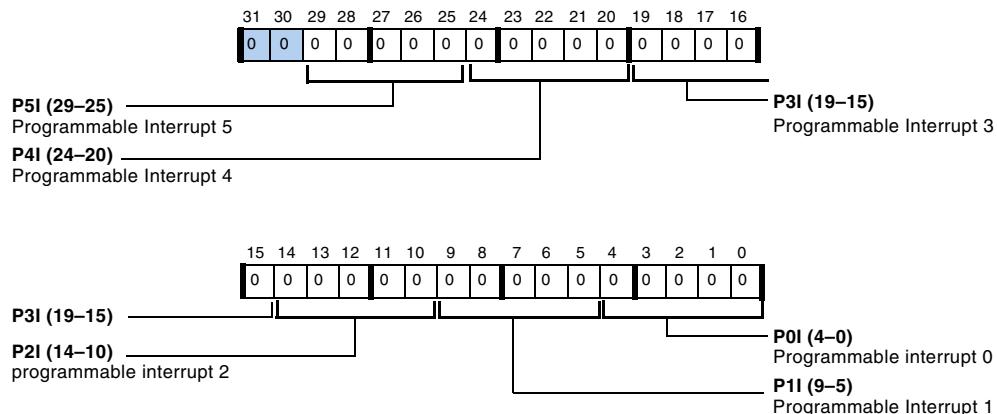


Figure B-1. PICR0 Register

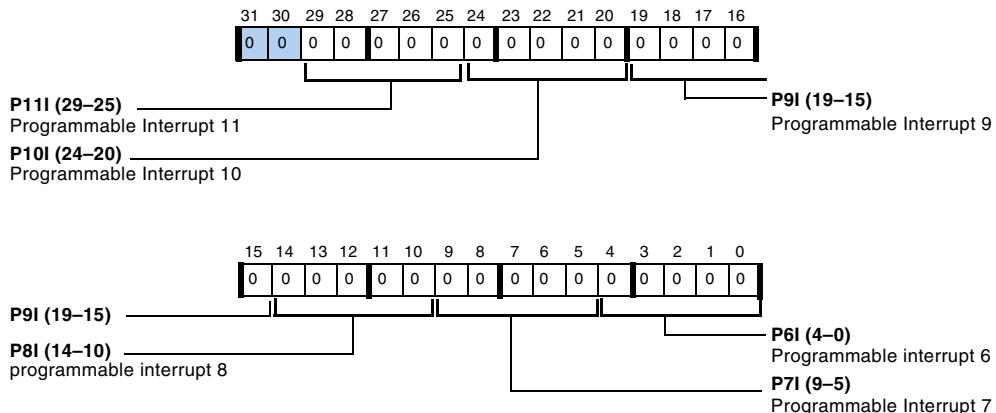


Figure B-2. PICR1 Register

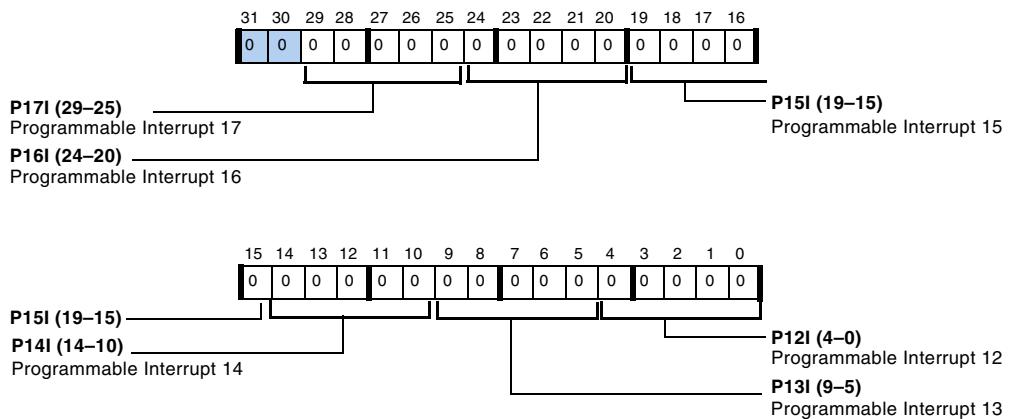


Figure B-3. PICR2 Register

Interrupt Priority

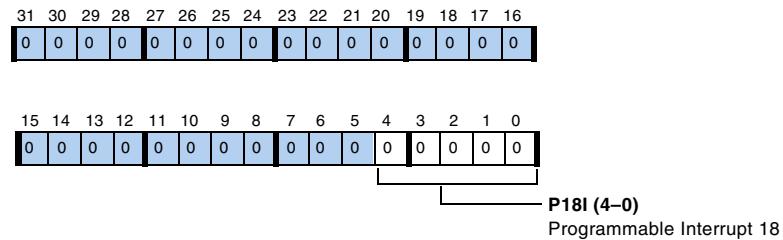


Figure B-4. PICR3 Register

C AUDIO FRAME FORMATS

This appendix introduces all the serial timing protocols used for audio inter-chip communications. These formats are listed and their availability in the various peripherals noted in [Table C-1](#).

Table C-1. Audio Format Availability

Frame Format	SPORTs	IDP/SIP	ASRC Input	ASRC Output	S/PDIF Tx	S/PDIF Rx	PCG
Serial	Yes						Yes
I ² S	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Left-justified	Yes	Yes	Yes	Yes	Yes		Yes
Right-justified, 24-bit		Yes	Yes	Yes	Yes		Yes
Right-justified, 20-bit		Yes	Yes	Yes	Yes		Yes
Right-justified, 18-bit		Yes	Yes	Yes	Yes		Yes
Right-justified, 16-bit		Yes	Yes	Yes	Yes		Yes
TDM, 128 channel	Yes		Yes	Yes			Yes

Overview

The following protocols are available in the SHARC processor and are briefly described in this appendix. For complete information on the industry standard protocols, see the specification listings in each section.

- Standard Serial Mode
- Left-justified Mode (Sony format)
- I²S Mode (Sony/Philips format)
- Time Division Multiplex (TDM) Mode
- MOST Mode
- Right-justified Mode
- S/PDIF (consumer mode)
- EBU/AES3 (professional mode)

Standard Serial Mode

Most processors allow word lengths of 4 to 32 bits to be transmitted or received through their serial ports. For convenience, most AFE (analog front-end) devices operate with 16-bit word lengths for both data and status transfer between the AFE and processor. The serial ports (SPORTs) of most DSPs are designed for full-duplex operation. They differ from the typical serial interface of micro controllers in that they use a frame sync pulse to indicate the start of the data frame. In the case of full duplex asynchronous transfers, two separate FS pulses are used for transmit and receive. The typical micro controller serial interfaces use the serial clock (`SCLK`) as an indicator of serial data, meaning that the `SCLK` is only active when data is valid. The DSP serial interface can operate with a continuous

SCLK, in which the frame synchronization (FS) pulse indicates the start of valid data.

Serial mode allows a flexible timing which can be used in unframed mode or framed mode. In framed mode the user can select between timing for early and late frame sync. Moreover the word order can be selected as LSB or MSB first.

I²S Mode

The Inter-IC-Sound (I²S) bus protocol is a popular 3 wire serial bus standard that was developed to standardize communication across a wide range of peripheral devices. Today the I²S protocol has become the standard method of communicating with consumer and professional audio products.

The I²S protocol provides transmission of 2 channel (stereo) Pulse Code Modulation digital data, where each audio sample is sent MSB first. The following list shows applications that use this format.

- Audio D/A and A/D converters
- PC multimedia audio controllers
- Digital audio transmitters and receivers that support serial digital audio transmission standards, such as AES/EBU, S/PDIF, IEC958, CP-340, and CP-1201
- Digital audio signal processors
- Dedicated digital filter chips
- Sample rate converters



Timing diagrams for I²S, right-justified and left-justified formats can be found in the product specific data sheet.

I²S Mode

The I²S bus transmits audio data from 8–32 bits and control signals over separate lines. The data line carries two multiplexed data channels—the left channel and the right channel. In I²S mode, if both channels on a SPORT are set up to transmit, then the SPORT transmits left and right I²S channels simultaneously. If both channels on a SPORT are set up to receive, the SPORT receives left and right I²S channels simultaneously. Data is transmitted in MSB-first format.

I²S consists, as stated above, of a bit clock, a word select and the data line. The bit clock pulses once for each discrete bit of data on the data lines. The bit clock operates at a frequency which is a multiple of the sample rate. The bit clock frequency multiplier depends on number of bits per channel, times the number of channels. For example, CD Audio with a sample frequency of 44.1 kHz and 32 bits of precision per (2) stereo channels has a bit clock frequency of 2.8224 MHz. The word select clock lets the device know whether channel 1 or channel 2 is currently being sent, since I²S allows two channels to be sent on the same data line.

Transitions on the word select clock also serve as a start-of-word indicator. The word clock line pulses once per sample, so while the bit clock runs at some multiple of the sample frequency, the word clock always matches the sample frequency. For a two channel (stereo) system, the word clock is a square wave, with an equal number of bit clock pulses clocking the data to each channel. In a mono system, the word clock pulses one bit clock length to signal the start of the next word, but is no longer be square. Instead, bit clocking transitions occur with the word clock either high or low.

Note the major difference between I²S and left/right justified modes is a left MSB data shift by one SCLK cycle in relation to the frame.

Standard I²S data is sent from MSB to LSB, starting at the left edge of the word select clock, with one bit clock delay. This allows both the transmitting and receiving devices to ignore the audio precision of the remote

device. If the transmitter is sending 32 bits per channel to a device with only 24 bits of internal precision, the receiver ignores the extra bits of precision by not storing the bits past the 24th bit. Likewise, if the transmitter is sending 16 bits per channel to a receiving device with 24 bits of precision, the receiver zero-fills the missing bits. This feature makes it possible to mix and match components of varying precision without reconfiguration.

Left-Justified Mode

Left-justified mode (also known as SONY Format) is a mode where in each frame sync cycle two samples of data are transmitted/received—one sample on the high segment of the frame sync, the other on the low segment of the frame sync. Prior to development of the 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.

Right-Justified Mode

Right-justified mode is a mode where in each frame sync cycle two samples of data are transmitted/received—one sample on the high segment of the frame sync, the other on the low segment of the frame sync. Prior to development of the 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.

TDM Mode

Programs have control over various attributes of this mode. One attribute is the number of bits (8- to 32-bit word lengths). However, each sample of the pair that occurs on each frame sync must be the same length.

TDM Mode

Many applications require multiple I/O channels to implement the desired system functions (such as telephone line and acoustic interfaces). Because most DSPs provide one, or at most two SPORTs, and one of these may be required for interfacing to the host or supervisory processor, it may be impractical, if not impossible, to dedicate a separate SPORT interface to each AFE connection.

The solution is to devise a way to connect a series of serial devices to one SPORT. Different converter manufacturers have approached this task in different ways. In essence, though, there are only two choices; either a time division multiplexing (TDM) approach, where each device is active on the SPORT in a particular time slot, or a cascading approach, where all devices are daisy chained together and data is transferred by shifting it through the chain and then following with a latching signal or a serial protocol. [Figure C-1](#) illustrates a pulsed frame clock for the TDM operation.

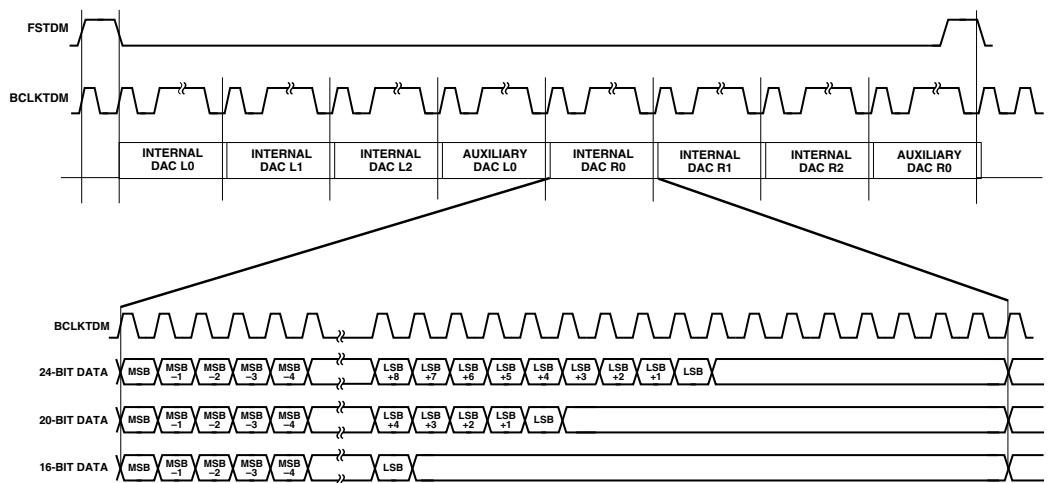


Figure C-1. TDM Mode Timing

Packed I²S Mode

This mode allows applications to send more than the standard 32 bits per channel available through standard I²S mode. Packed mode is implemented using standard TDM mode. Packed mode supports up to 128 channels as does TDM mode as well as the maximum of (128 x 32) bits per left or right channel. As shown in [Figure C-2](#) packed I²S waveforms are the same as the wave forms used in TDM mode, except that the frame sync is toggled for every frame, and therefore emulates I²S mode. So it is a hybrid between TDM and I²S mode.

TDM Mode

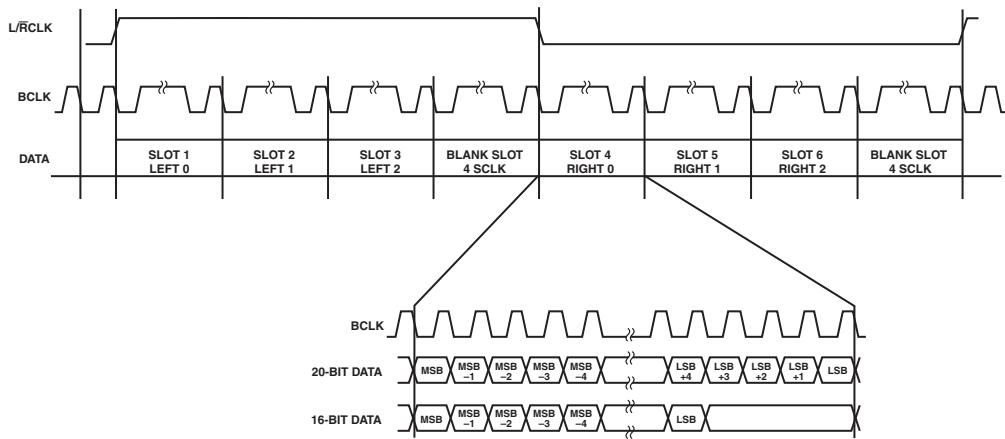


Figure C-2. Packed I2S Mode

MOST Mode

A special packed TDM mode is available that allows four channels to be fit into a space of 64-bit clock cycles. This mode is called packed TDM4 mode, or MOST™ mode. MOST (Media Oriented Systems Transport) is a networking standard intended for interconnecting multimedia components in automobiles and other vehicles. Many integrated circuits intended to interface with a MOST bus use a packed TDM4 data format.

Figure C-3 illustrates a word length of 16 bits for a timing diagram of the packed TDM4 mode. This figure is shown with a negative **BCLK** polarity, a negative **LRCLK** polarity, and an MSB delay of 1. The MSB position of the serial data must be delayed by one bit clock from the start of the frame (I^2S position).

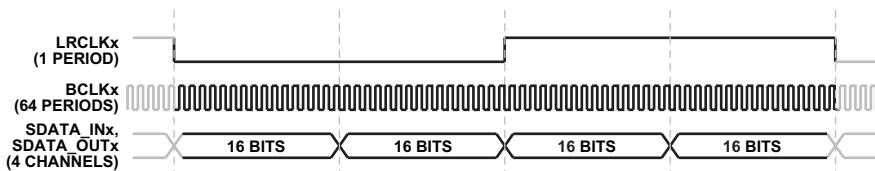


Figure C-3. Packed TDM4 Mode

AES/EBU/SPDIF Formats

For this section, it is important to be familiar with serial digital application interface standards IEC-60958, EIAJ CP-340, AES3 and AES11.

S/PDIF data is transmitted as a stream of 32-bit data words. A data frame consists of 384 words in total, with 192 data words transmitted for the A stereo channel, and 192 data words transmitted for the B stereo channel.

The difference between the AES/EBU and S/PDIF protocol is the channel status bit. If the channel status bit is not set, then:

- 0 = Consumer/professional
- 1 = Normal/compressed data
- 2 = Copy prohibit/copy permit
- 3 = 2 channels/4 channels
- 4 = n/a
- 5 = No pre-emphasis/pre-emphasis

AES/EBU/SPDIF Formats

There is one channel status bit in each sub-frame, (comprising of 192 bits per audio block). This translates to $192/8 = 24$ bytes available (per audio block). The meaning of the channel status bits are as follows

- The biphase encoded AES3 stream is composed of subframes ([Figure C-5 on page C-13](#)). Subframes consist of a preamble, four auxiliary bits, a 20-bit audio word, a validity bit, a user bit, a channel status bit, and a parity bit.
- The preamble indicates the start of the subframe. The four auxiliary bits normally are the least significant bits of the 24-bit audio word when pasted to the 20-bit audio word. In some cases, the auxiliary bits are used to convey some kind of other data indicated by the channel status bits.
- The validity bit (if cleared, =0) indicates the audio sample word is suitable for direct analog conversion. User data bits may be used in any way desired by the program. The channel status bit conveys information about the status of the channel. Examples of status are length of audio sample words, number of audio channels, sampling frequency, sample address code, alphanumeric source, and destination codes and emphasis. The parity bit is set or cleared to provide an even number of ones and of zeros for time slots 4-31.
- Each frame in the AES3 stream is made up of two subframes. The first subframe is channel A, and the second subframe is channel B. A block is comprised of 192 frames. The channel status is organized into two 192 bit blocks, one for channel A and one for channel B. Normally, the channel status of channel A is equal to channel B. It is extremely rare that they are ever different. Three different preambles are used to indicate the start of a block and the start of channel A or B.
 1. Preamble Z indicates the start of a block and the start of subframe channel A

2. Preamble X indicates the start of a channel A subframe when not at the start of a block.
3. Preamble Y indicates the start of a channel B subframe.

The user bits from the channel A and B subframes are simply strung together. For more information, please refer to the AES3 standard.

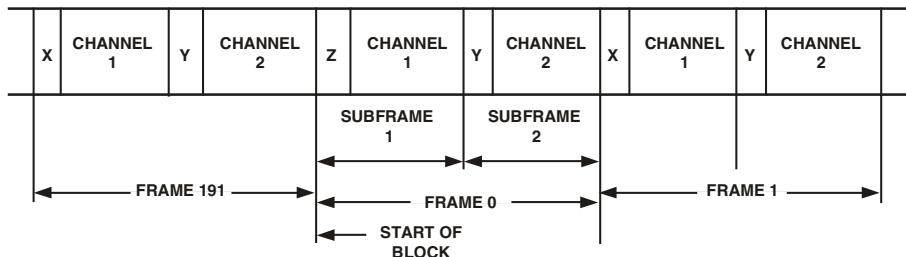


Figure C-4. S/PDIF Block Structure

The data carried by the 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 C-4). Since the time slots correspond with the data bits, the frame is often described as being 64 bits in length.

A frame is uniquely composed of two subframes. The first subframe normally starts with preamble X. However, the preamble changes to preamble Z once every 192 frames. This defines the block of frames structure used to organize the channel status information. The second subframe always starts with preamble Y.

¹ The unit interval is the minimum time interval between condition changes of a data transmission signal.

Subframe Format

Each frame consists of two subframes. [Figure C-5](#) shows an illustration of a subframe, which consists of 32 time slots numbered 0 to 31. A subframe is 64 unit intervals in length. The first four time slots of each subframe carry the preamble information. The preamble marks the subframe start and identifies the subframe type. The next 24 time slots carry the audio sample data, which is transmitted in a 24-bit word with the least significant bit (LSB) first. When a 20-bit coding range is sufficient, time slots 8 to 27 carry the audio sample word with the LSB in time slot 8. Time slots 4 to 7 may be used for other applications. Under these circumstances, the bits in time slots 4 to 7 are designated auxiliary sample bits. If the source provides fewer bits than the interface allows (either 20 or 24), the unused LSBs are set to logic 0.

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.

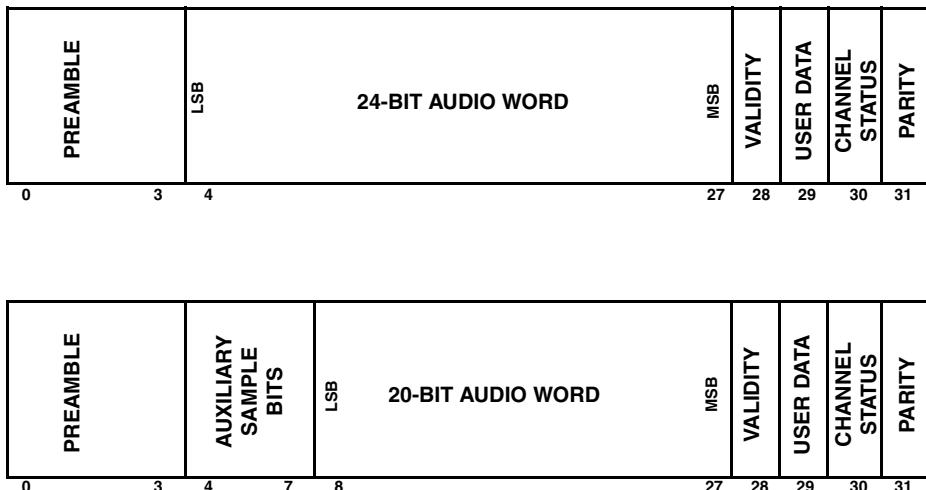


Figure C-5. Subframe Format

After the audio sample word, there are four final time slots which carry:

1. **Validity bit (time slot 28).** The validity bit is logic 0 if the audio sample word is suitable for conversion to an analog audio signal, and logic 1 if it is not. This bit is set if the CHST_BUF_ENABLE bit and the VALIDITY_A (VALIDITY_B for channel 2) bit is set in the SPDIF_TX_CTL register. This bit is also set if the corresponding bit given with the sample is set.
2. **User data bit (time slot 29).** This bit carries user-specified information that may be used in any way. This bit is set if the corresponding bit given with the left/right sample is set.
3. **Channel status bit (time slot 30).** The channel status for each audio signal carries information associated with that audio signal, making it possible for different channel status data to be carried in the two subframes of the digital audio signal. Examples of information to be carried in the channel status are: length of audio sample

words, number of audio channels, sampling frequency, sample address code, alphanumeric source and destination codes, and emphasis.

Channel status information is organized in 192-bit blocks, subdivided into 24 bytes. The first bit of each block is carried in the frame with preamble Z.

4. **Parity bit (time slot 31).** The parity bit indicates that time slots 4 to 31 inclusive will carry an even number of ones and an even number of zeros (even parity). The parity bit is automatically generated for each subframe and inserted into the encoded data.

The two subframes in a frame can be used to transmit two channels of data (channel 1 in subframe 1, channel 2 in subframe 2) with a sample rate equal to the frame rate. Alternatively, the two subframes can carry successive samples of the same channel of data, but at a sample rate that is twice the frame rate. This is called single-channel, double-frequency (SCDF). [For more information, see “Data Output Mode” on page 13-12.](#)

Channel Coding

To minimize the direct-current (dc) component on the transmission line, to facilitate clock recovery from the data stream, and to make the interface insensitive to the polarity of connections, time slots 4 to 31 are encoded in bi-phase mark.

Each bit to be transmitted is represented by a symbol comprising two consecutive binary states. The first state of a symbol is always different from the second state of the previous symbol. The second state of the symbol is identical to the first if the bit to be transmitted is logic 0. However, it is different if the bit is logic 1.

[Figure C-6](#) shows that the ones in the original data end up with mid cell transitions in the bi-phase mark encoded data, while zeros in the original

data do not. Note that the bi-phase mark encoded data always has a transition between bit boundaries.

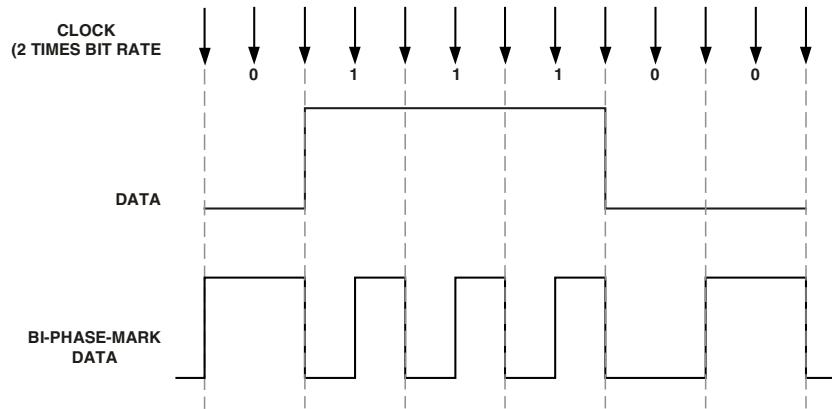


Figure C-6. Bi-phase Mark Encoding

Preambles

Preambles are specific patterns that provide synchronization and identify the subframes and blocks. To achieve synchronization within one sampling period and to make this process completely reliable, these patterns violate the bi-phase mark code rules, thereby avoiding the possibility of data imitating the preambles.

A set of three preambles, shown in [Table C-2](#), are used. These preambles are transmitted in the time allocated to four time slots at the start of each subframe (time slots 0 to 3) and are represented by eight successive states. The first state of the preamble is always different from the second state of the previous symbol (representing the parity bit).

AES/EBU/SPDIF Formats

Table C-2. Preambles

Preamble	Preceding state 0	Preceding state 1	Description
X	11100010	00011101	Subframe 1
Y	11100100	00011011	Subframe 2
Z	11101000	00010111	Subframe 1 and block start

Like bi-phase code, the preambles are dc free and provide clock recovery. They differ in at least two states from any valid bi-phase sequence.

I INDEX

Numerics

128-channel TDM, [10-3](#)
16-bit SDRAM address map, [3-28](#), [3-62](#)
16-bit word, [3-16](#), [10-45](#)
16-bit word, boot packing, [23-19](#)
32/40-bit floating-point mode, IIR
accelerator, [6-61](#)
32-bit word, [3-89](#), [3-117](#), [5-4](#), [10-40](#),
[10-41](#), [10-43](#), [11-6](#), [11-7](#), [11-10](#),
[11-12](#), [11-19](#)
32-bit word, boot packing, [23-18](#)
32- to 40-bit packing, IIR, [6-62](#)
48-bit word, [3-90](#)
48-bit word, boot packing, [23-15](#)
8-bit boot (SPI), [23-17](#)
8-bit word, [3-14](#), [A-49](#)
8-bit word, boot packing, [23-20](#)

A

AAC compressed format, [13-17](#)
AC-3 format, [13-17](#)
accuracy (PWM), [7-23](#)
active low frame sync select for frame sync
(INVFSx) bit, [14-13](#)
active state multichannel receive frame sync
select (LMFS) bit, [10-20](#)
AD1855 stereo DAC, use with SPI, [15-10](#)
address
AMI, [3-84](#)
AMI address map, [3-14](#)
core to external memory, [3-27](#)

address *(continued)*
decoding address bank, [3-27](#), [3-62](#)
destination, [2-22](#)
instruction execution from external
memory, [3-89](#)
internal/external index, [2-36](#)
logical vs. physical, [3-90](#)
map, 16-bit SDRAM, [3-28](#), [3-62](#)
predictive, [3-14](#)
SDRAM, [3-22](#), [3-24](#)
SDRAM read, [3-40](#)
addressing
7-bit in TWI, [21-2](#)
AMI, external, [3-14](#), [3-15](#)
byte in SDRAM, [3-68](#)
general call in TWI, [21-14](#)
IOP, [2-25](#)
mixing instructions and data, [3-93](#)
alarm clock, RTC, [18-7](#)
AMI
See also external port, SDRAM, shared
memory
ADDR23-0 bits, [3-14](#)
address map, [3-14](#)
control (AMICTLx) register, [A-21](#) to
[A-23](#), [A-47](#) to [A-50](#)
DMA, [3-10](#)
external memory addressing, [3-14](#), [3-15](#)
memory bank support, [3-14](#)
reading external memory, [3-84](#)
read/write throughput, [3-117](#)

Index

- AMI *(continued)*
status (AMISTAT) register, A-23, A-24, A-51
timing, 3-7
- AMI bits
ACK pin enable (ACKEN), A-22, A-49
buffer flush (FLSH), A-23, A-50
bus hold cycle (HC), A-22, A-49
bus idle cycle (HC), A-23, A-50
external bus data width (BW), A-22
most significant word first (MSWF), A-22, A-49
packing disable (PKDIS), A-22, A-49
predictive read disable (NO_OPT), A-23, A-50
read hold cycle (RHC), A-23, A-50
wait state enable (WS), A-22, A-49
- AMIEN, A-22, A-48
- arbitration, channel, 2-36
- arbitration, fixed/floating, 2-43
- architecture
 TWI controller, 21-7
- array, hold TCB, 2-35
- asynchronous memory interface. *See* AMI
- asynchronous serial communications
 (UART), 20-7
- audience, intended, -lxi
- audio
 biphase encoded in S/PDIF, 13-3
 data formats, IDP, 11-16
 formats, IDP, 11-6
 formats, S/PDIF, 13-7, 13-15
 non-linear data, S/PDIF, 13-16
 restriction with SPORTs, 10-10
 transmission standards, SPORTs, C-3
- audio formats, C-1 to C-9
- audio modes, C-2 to C-9
- autobaud detection, 20-6
- automotive products, 1-2
- B
bank
 DDR2 address mapping, 3-65, 3-66
 SDRAM address mapping, 3-25 to 3-30
- base registers, 2-8
- baud rate, 23-13
 setting, 15-30
 UART, 20-5, 20-10, 20-11
- BHD (buffer hang disable) bit, 10-54
- bidirectional connections through the signal routing unit, 9-14
- bidirectional functions (transmit, receive), 10-3
- biphase
 encoded audio stream, 13-4, 13-14
 routing data, 13-5
- bits *See* peripheral specific bits, bits by name or acronym
- block diagram
 FFT accelerator, 6-6
 IDP, 11-7
 IDP channel 0, 11-9
 I/O processor, 2-30
 PWM, 7-2
 S/PDIF transmitter, 13-8
 SPI, 15-9
 SPORTs, 10-13
 SRC, 12-6
 TWI controller, 21-7
- block diagrams
 RTC, 18-8
- boolean operator
 OR, 11-32
- booting, 23-7 to 23-28
 bootstrap loading, 23-7
 DMA use in, 2-23
 link port, 23-21
 SPI master mode, 23-12
 SPI packing, 23-17
 SPI slave, 23-15

booting *(continued)*

- SPI slave mode, [23-15](#)

buffer

- addressing, [2-25](#)
- data, [2-10](#)
- DMA use in, [2-24](#)
- SPORT data, [10-1](#)
- TCB allocation, [2-33](#)

buffer, pin, [9-7](#)

buffer hang disable (BHD) bit, [10-54](#), [A-156](#), [A-160](#), [A-165](#)

buses

- errors in, [3-44](#), [3-80](#)
- external bus data width (BW) bit, [A-22](#), [A-48](#)
- external port, [2-43](#)
- hold cycle bit, [A-22](#), [A-49](#)
- I²S and, [C-3](#)
- idle cycle bit, [A-23](#), [A-50](#)
- I/O address (IOA), [2-25](#)
- IO data (IOD), [2-30](#)
- I/O processor (IOP), [10-47](#)
- peripheral, [2-44](#)

bypass as a one-shot (strobe pulse), [14-13](#)

C

capacitors

- bypass, [23-36](#)
- decoupling, [23-36](#)

CAS latency

- bit (SDCL), [A-53](#)

cautions and warnings

- DMA transfers, [2-28](#)
- I/O processor, [2-28](#)
- SPORTs, [10-42](#)

center-aligned paired PWM

- double-update mode, [7-10](#)
- single-update mode, [7-9](#)

chain assignment, I/O processor, [2-33](#)

chained DMA, [2-23](#), [2-29](#), [2-32](#), [2-34](#)

- chained DMA enable (SCHEN_A and SCHEN_B) bit, [A-156](#), [A-160](#)
- chained DMA sequences, [2-11](#)
- chain pointer (CPSPI) registers, SPI, [15-32](#)
- chain pointer (CPSPx) registers, SPORTs, [2-12](#), [10-47](#)
- chain pointer (CPx) registers, [2-33](#)
- chain pointer registers (general), [2-11](#)
- enable (SCHEN_A and SCHEN_B) bit, [10-47](#), [A-164](#)
- FFT accelerator, [2-18](#)
- FIR accelerator, [2-16](#)
- IIR accelerator, [2-17](#)
- link port, [2-15](#)
- SPI, [2-14](#)
- SPORTs, [2-14](#), [10-47](#), [A-156](#), [A-160](#), [A-164](#)
- UART, [2-15](#), [20-14](#)

chaining enable bit (CHEN), [2-34](#)

chain insertion mode, SPORT, [10-48](#)

chain pointer registers, [2-7](#), [2-11](#)

changing SPI configuration, [15-29](#)

channel

- allocation for DMA, [2-3](#)
- arbitration, fixed/floating, [2-43](#)
- defined, [2-2](#)
- DMA, [2-23](#)
- granting access, [2-31](#)
- interrupt, [2-45](#), [2-48](#)
- priority, [2-31](#)
- priority for DMA, [2-30](#)
- registers, listed, [2-36](#)

channel B transmit status register (SPDIF_TX_CHSTB), [A-203](#), [A-204](#)

channel selection registers, [10-36](#)

clearing interrupts, latches, [B-3](#)

CLKOUTEN (clockout enable) bit, [A-11](#), [A-16](#)

Index

- clock A source (CLKASOURCE) bit, [A-193](#)
clock input (CLKIN) pin, [17-5](#)
clocks
 RTC, [18-3](#)
clocks and system clocking, [22-2](#)
 clock and frame sync frequencies (DIVx)
 registers, [10-8](#)
 clock distribution, [23-33](#)
 clock polarity (CLKPL) bit, [A-234](#)
 clock rising edge select (CKRE) bit, [A-155, A-164](#)
core clock, [22-6](#)
disabling the clock, [22-6](#)
hardware control, [22-4](#)
internal clock select (ICLK) bit, [A-154, A-159, A-163](#)
 managing for power savings, [22-12](#)
 output divider, [22-4](#)
 peripheral clock, [22-6](#)
 precision clock generator registers, [14-21](#)
SDRAM controller, [3-6](#)
selecting clock ratios, [22-4](#)
software control, [22-4](#)
source select (MSTR) bit, [A-154, A-159, A-163](#)
SPI clock phase select (CPHASE) bit, [A-234](#)
SPORTs, [10-8](#)
VCO encodings, [22-5](#)
coefficient memory, FIR, [6-33](#)
commands
 auto-refresh, [3-24](#)
 bank activate, [3-21](#)
 load mode register, [3-21](#)
 NOP, [3-24](#)
 precharge, [3-21, 3-22](#)
 read/write, [3-22](#)
 self-refresh, [3-44, 3-79](#)
 compand data in place, [10-3](#)
companding (compressing/expanding), [10-3](#)
compute block, FFT, [6-5](#)
conditioning input signals, [23-33](#)
configuring frame sync signals, [10-11](#)
connecting peripherals through DAI, [9-16](#)
connections
 group A, clock signals, [9-24](#)
 group A, DPI, input routing signals, [A-218](#)
 group B, DPI, pin assignment signals, [A-223](#)
 group C, DPI, pin enable signals, [A-227](#)
continuous mode, *See* SPORTs, framed
 and unframed data
controller, SDRAM, [3-17](#)
conventions, [-lx_x](#)
core access read optimization, [3-42, 3-78](#)
core address mapping, [3-27](#)
core transmit/receive operations, [15-20](#)
count (CSPx) DMA registers, [2-6](#)
count (CSPx) registers, [2-27](#)
counters
 RTC, [18-4](#)
count (IDP_DMA_Cx) registers, [11-20, 11-21](#)
crosstalk, reducing, [23-37](#)
CSPx (peripheral DMA counter) registers, [2-6, 2-27](#)
customer support, [-lxvi](#)

D

- DAI
 clock routing control registers (group A), [9-24](#)
 configuration macro, [9-43](#)
 connecting peripherals with, [9-16](#)
 control registers, clock routing control
 registers (Group A), [A-118](#)

DAI (continued)
DAI interrupt falling edge
(DAI_IRPTL_FE) register, 11-27
DAI interrupt rising edge
(DAI_IRPTL_RE) register, 11-27
DAI_IRPTL_FE register
as replacement to IMASK, 9-39
DAI_IRPTL_H register, 11-23
DAI_IRPTL_H register as replacement
to IRPTL, 9-39
DAI_IRPTL_L register as replacement
to IRPTL, 9-39
DAI_IRPTL_PRI register, 9-39, 11-27
DAI_PIN_STAT register, A-148
DAI_STAT register, 11-22, A-182,
A-183
edge-related interrupts, 9-37
interrupt controller, 9-32 to 9-38
interrupt controller registers, A-149
interrupts, 9-33
ping-pong DMA status
(SRU_PINGx_STAT) register,
A-182, A-183
pin status (DAI_PIN_STAT) register,
A-148
routing, 9-20
rules for routing, 9-20
selection group E (miscellaneous signals),
A-140
SPORT SRU signal connections, 10-5
status (DAI_STAT) register, A-182,
A-183
system configuration, sample, 9-42
system design, 9-4
DAI_IRPTL_RE register
as replacement to IMASK, 9-39
DAI registers
pin status (DAI_PIN_STAT), A-230

data
 buffer, FIR, [6-33](#)
 direction control (SPTRAN) bit, [A-157](#),
 [A-160](#), [A-165](#)
 type select (DTYPE) bit, [A-153](#), [A-163](#)
data-independent frame sync, [10-20](#)
 (DIFS) mode, [10-20](#)
data memory, FFT, [6-6](#)
data ready (DR) status flag (UART), [20-11](#)
data type
 and companding, [10-14](#)
 and formatting (non-multichannel),
 [10-22](#)
data words
 single word transfers, [10-43](#)
 transferring, [10-26](#), [10-36](#)
 UART, [20-10](#)
DDR2, [3-46](#) to [3-83](#)
 16-bit address mapping, [3-63](#) to [3-66](#)
 addressing (16-bit, interleaving), [3-65](#)
 bank address, [3-27](#), [3-62](#)
 commands, [3-51](#)
 DDR2 DLL description, [3-48](#)
 decoding address bank, [3-27](#), [3-62](#)
 delay generation, [3-50](#)
 interleaving, [3-60](#)
 latency, [3-52](#), [3-54](#), [3-55](#), [3-129](#)
 memory chip select pins (DDR2_CSx),
 [3-46](#)
 throughput, [3-10](#), [3-118](#)
DDR2 bits
 address mode (ADDRMODE), [3-60](#)
 auto refresh (DDR2ORF), [A-28](#), [A-43](#)
 bank count, 4 or 8 (DDR2BC), [A-26](#),
 [A-41](#), [A-42](#)
 column address width (DDR2CAW),
 [3-62](#), [A-26](#)
 disable access (DIS_DDCTL), [A-26](#)
 disable clock and control
 (DIS_DDCLK1), [A-26](#)

Index

- DDR2 bits *(continued)*
- disable DDCKE signal (DIS_DDCKE), [A-26](#)
 - disable SHARC DLL (SH_DLL_DIS), [A-26](#)
 - exit self refresh (SREF_EXIT), [A-28](#)
 - external data path width (X16DE), [A-27](#), [A-41](#), [A-42](#), [A-43](#), [A-44](#)
 - force auto refresh (FARF), [A-28](#)
 - force DLL calibration (ForceDLLcal), [A-27](#)
 - force EMR2 register write (FEMR2), [A-27](#)
 - force EMR3 register write (FEMR3), [A-27](#)
 - force EMR register write (FEMR), [A-28](#)
 - force MR register write (FLMR), [A-28](#)
 - force precharge (Force PC), [A-28](#)
 - pipeline enable (DDR2BUF), [A-28](#)
 - power-up sequence start (DDR2PSS), [A-27](#)
 - read modify (DDR2MODIFY), [A-29](#)
 - read optimization enable (DDR2OPT), [A-29](#)
 - row address width (DDR2RAW), [3-62](#), [A-27](#), [A-43](#)
 - self refresh mode (DDR2SRF), [A-28](#)
- DDR2 commands
- force load mode register, [3-52](#)
- DDR2 registers
- control (DDR2CTL0), [3-51](#), [3-57](#), [3-60](#), [A-25](#)
- dead time example (PWM), [7-16](#)
- debug, [10-60](#), [23-33](#)
- DAI use in, [9-10](#), [9-40](#)
 - data buffer use in, [2-11](#)
 - FIR, [6-45](#)
 - SPI, [4-21](#), [15-27](#), [15-28](#)
 - SPORT, [10-53](#)
 - tools use in, [1-5](#)
- decimation, FIR, [6-38](#)
- destination address, [2-22](#)
- DIFS (data independent frame sync select) bit, [A-155](#), [A-159](#)
- digital applications interface. *See* DAI
- digital loopback mode, *See* peripheral specific loopback mode
- digital peripheral interface, *See* DPI
- DIVEN (PLL divider enable) bit, [22-15](#), [A-8](#), [A-10](#), [A-14](#), [A-16](#)
- divisor, UART, [20-4](#), [A-249](#)
- divisor (DIVx) registers, [10-8](#), [10-11](#)
- DIVx (divisor) registers, [A-151](#)
- DLAB (divisor latch access) bit, [A-243](#)
- DMA
- access, granting, [2-31](#)
 - arbitration, [2-44](#)
 - base registers, [2-8](#)
 - booting, [2-23](#)
 - bus priority, [2-42](#)
 - chained, FFT, [2-18](#)
 - chained, FIR, [2-16](#)
 - chained, IIR, [2-17](#)
 - chained, link port, [2-15](#)
 - chained, SPI, [2-14](#)
 - chained, SPORT, [2-14](#)
 - chained, UART, [2-15](#), [20-14](#)
 - chaining, [2-23](#), [2-32](#) to [2-36](#)
 - chaining enable bit (CHEN), [2-34](#)
 - chain insertion mode, [2-36](#)
 - chain loading priority, [2-35](#)
 - chain pointer registers, [2-7](#), [2-11](#)
 - channel, buffer registers, listed, [2-36](#)
 - channel, parameter registers, [2-36](#)
 - channel allocation, [2-3](#), [2-36](#)
 - channel arbitration, [2-36](#)
 - channel paths, [2-25](#)
 - channel priority, [2-30](#)
 - channel priority, external port, [2-43](#), [A-20](#), [A-46](#)

DMA *(continued)*

- channels, [2-2](#)
- circular buffered, [2-8](#), [2-24](#)
- configuring in the I/O processor, [2-51](#)
- control/status registers, [2-36](#)
- data buffer, [2-32](#), [2-36](#)
- direction change (external port), [2-23](#)
- extended parameter registers, [2-8](#)
- external port bus, [2-43](#)
- external port bus priority, [A-20](#), [A-46](#)
- FFT chaining, [2-18](#)
- handshake, [2-30](#)
- IIR chaining, [2-17](#)
- index addresses, [2-27](#)
- index registers, [2-4](#)
- interrupts, [2-45](#) to [2-48](#)
- interrupt sources, FIR, [6-45](#), [6-65](#)
- loading chain, [2-34](#)
- miscellaneous external port parameter registers, [2-9](#)
- offset value, [2-27](#)
- operation, master mode, [15-31](#), [15-33](#)
- parameter registers, [2-36](#)
- ping-pong, [2-23](#), [2-24](#)
- ping-pong, IDP, [11-21](#)
- ping-pong enable (IDP_PING) bits, [A-178](#)
- program controlled interrupt (PCI) bit, [2-12](#), [2-13](#), [6-18](#), [10-48](#), [15-13](#), [20-22](#), [A-167](#)
- read optimization example, [3-43](#), [3-78](#)
- rotating peripheral priority, [2-44](#)
- slave, interrupts, [4-19](#)
- SPI slave mode, [15-31](#), [15-32](#)
- SPORT chain status bits (DMACHSxy), [A-171](#)
- SPORT status bit (DMASSxy), [A-171](#)
- standard (non chained), [2-23](#)
- starting/stopping, [2-29](#)

DMA *(continued)*

- switching from receive to transmit mode, [15-35](#)
- switching from transmit to receive mode, [15-34](#)
- TCB, [2-21](#), [2-32](#)
- TCB allocation, [2-32](#)
- TCB size, [2-32](#)
- transfer control block See DMA TCB
- transfer direction, external port, [2-24](#)
- transfers, [11-20](#)
- DMACx (external port DMA registers), [A-60](#)
- DMA example, chain assignment, [2-34](#)
- Dolby, DTS audio standards (S/PDIF), [13-13](#)
- DPI

 - connections, group A, [A-218](#) to [A-222](#)
 - connections, group B, [A-223](#) to [A-226](#)
 - connections, group C, [A-226](#) to [A-230](#)
 - pin assignment (group B) signals, [A-223](#)
 - pin enable (group C) signals, [A-226](#)

- DSP, architectural overview, [1-2](#)
- DSxEN (SPI device select) bits, [15-31](#), [15-33](#), [A-242](#)
- DTS format, [13-17](#)
- DTYPE, [10-22](#)
- DTYPE (data type) bits, [A-153](#), [A-163](#)
- DXS_B, DSX_A (data buffer channel A/B status) bit, [A-157](#), [A-158](#), [A-161](#), [A-166](#)

E

- early vs. late frame syncs, [10-27](#)
- edge-related interrupts, [9-37](#)
- ELSI (enable RX status interrupt) bit, [A-247](#)
- emergency dead time example, [7-16](#)

Index

- enable
- DMA interrupt (INTEN) bit, [15-26](#)
 - EXT_CLK mode, [16-15](#)
 - external clock synchronization, PCG, [14-20](#)
 - external port (asynchronous memory interface), [A-22](#), [A-48](#)
 - multichannel mode in SPORTs, [A-170](#)
 - PCGs, [14-19](#)
 - peripheral timer, [16-4](#), [16-21](#)
 - pin buffer, timer, [16-3](#)
 - pulse width modulation groups, [7-20](#)
 - PWM_OUT mode, [16-8](#)
 - SPIIDS (ISSEN) bit, [15-26](#)
 - synchronize (counter) bits, PWM, [7-25](#)
 - WDTH_CAP mode, [16-12](#)
- enable receive buffer full interrupt (ERBFI) bit, [A-246](#)
- enable transmit buffer empty interrupt (ETBEI) bit, [A-246](#)
- endian format, [15-2](#), [A-153](#)
- equation
- duty cycles in PWM, [7-10](#)
 - FIR throughput, [6-47](#)
 - frame sync frequency, [10-9](#)
 - frame sync pulse (SPORT), [10-9](#)
 - peripheral timer period, [16-11](#)
 - pulse width modulation switching frequency, [7-6](#)
 - SDRAM clock, [3-34](#)
 - SDRAM refresh rate, [3-33](#)
 - serial clock frequency, [10-9](#)
 - serial port clock divisor, [10-9](#)
 - SPI clock baud rate, [A-239](#)
- errors
- internal bus (SDRAM), [3-44](#), [3-80](#)
 - SPORT error control register, [A-172](#)
 - TWI master mode, [21-21](#), [21-23](#)
 - TWI repeat start, [21-24](#)
 - TWI slave transfer, [21-11](#), [21-20](#)
- errors *(continued)*
- UART baud rate, [20-5](#)
 - UART sampling, [20-11](#)
 - event flags, RTC, [18-14](#)
 - examples
 - bypass mode (PLL), [22-18](#)
 - capacitor placement, [23-36](#)
 - chain assignment, DMA, [2-34](#)
 - clock post divider, [22-14](#)
 - edge-aligned PWM, [7-18](#)
 - external port DMA read, [3-43](#), [3-78](#)
 - FIR filter loop, [3-97](#)
 - multibank SDRAM with data packing, [3-36](#), [3-73](#) - multiple processor system, [3-39](#), [3-76](#)
 - pin buffer, [9-7](#)
 - power management, [22-12](#)
 - PWM deadtime, [7-12](#)
 - PWM emergency dead time, [7-16](#)
 - PWM switching frequencies, [7-7](#)
 - read optimization, [3-42](#), [3-78](#)
 - reset generator, [23-42](#)
 - SDRAM clock, [3-33](#)
 - single processor system, [3-39](#), [3-76](#)
 - SPI interface with AD1855 DAC, [15-10](#)
 - SRU connections, [9-20](#) to [9-24](#)
 - timing for a SPORT multichannel transfer, [10-34](#)
 - token passing, [4-11](#)
 - VCO, [22-15](#)
- examples, timing
- link port handshake, [4-7](#)
 - SPI clock, [15-17](#)
 - SPI transfer protocol, [15-15](#)
 - SPORT framed vs. unframed data, [10-28](#)
 - SPORT normal vs. alternate framing, [10-28](#)
- exception interrupt, [9-32](#)
- EXT_CLK (external event watchdog) mode, [16-5](#)

- extended parameter registers, DMA, [2-8](#)
 external event watchdog (EXT_CLK)
 mode, [16-2](#), [16-15](#)
 external memory
 access restrictions, [3-132](#)
 access timing, [3-5](#)
 address bank decoding, [3-27](#), [3-62](#)
 addressing, AMI, [3-14](#), [3-15](#)
 banks, [3-17](#), [3-35](#)
 boot-up code for interrupt vector tables, [3-89](#)
 executing instructions from, [3-89](#)
 external physical address, [3-14](#)
 interface, [3-10](#)
 most significant word first (MSWF) bit, [A-22](#), [A-49](#)
 packing and unpacking data (PKDIS)
 bits, [A-22](#), [A-49](#)
 pin descriptions, [3-48](#)
 reads, [3-84](#)
 restrictions, access, [3-132](#)
 SDRAM, [3-35](#)
 select signals (MSx), [3-35](#)
 SPORT data transfers, [10-39](#)
 writes, [3-84](#)
 external port
 bus, [2-43](#)
 bus hold cycle bit, [A-22](#), [A-49](#)
 bus idle cycle bit, [A-23](#), [A-50](#)
 bus priority, [A-20](#), [A-46](#)
 channel freezing, [3-10](#)
 channel priority, [2-43](#), [A-20](#), [A-46](#)
 core address mapping, [3-27](#)
 data pin mode select (EPDATA) bits, [A-6](#)
 DMA bus, [2-43](#)
 DMA read optimization, [3-43](#), [3-78](#)
 DMA registers, [2-28](#), [A-60](#) to [A-63](#)
 external code throughput, [3-119](#)
 external index addressing, [2-28](#)
- external port *(continued)*
 feature summary, [4-2](#)
 instruction packing, [3-90](#)
 program controlled interrupt bit (PCI), [6-18](#)
 read hold cycle (RHC) bits, [A-23](#), [A-50](#)
 TCB, [2-19](#)
 transfer direction, DMA, [2-24](#)
 external port bits
 bank select (BxSD), [A-19](#), [A-46](#)
 bus priority (EPBR), [A-20](#), [A-46](#)
 data enable (DATA), [A-47](#)
 delay line write pointer write back status (WBS), [A-62](#)
 DMA chaining enable (CHEN), [A-61](#)
 DMA chain status (CHS), [A-62](#)
 DMA circular buffer enable (CBEN), [A-61](#)
 DMA delay-line enable (DLEN), [A-61](#)
 DMA direction (TRAN), [A-61](#)
 DMA enable (DMAEN), [A-61](#)
 DMA external interface status (EXTS), [A-62](#)
 DMA FIFO status (DFS), [A-62](#)
 DMA flush FIFO (DFLSH), [A-61](#)
 DMA transfer direction status (DIRS), [A-62](#), [A-63](#)
 DMA transfer status (DMAS), [A-62](#)
 freeze length core (FRZCR), [A-20](#), [A-21](#), [A-47](#)
 freeze length (FRZDMA), [A-20](#), [A-46](#)
 freeze (NOFRZDMA, NOFRZCR), [3-10](#)
 internal DMA complete interrupt (INIRT), [A-62](#)
 on the fly control loading enable (OFCEN), [A-61](#)
 tap list DMA enable (TLEN), [A-62](#)
 write back of EPEI after reads/writes (WRBEN), [A-61](#)

Index

external port DMA direction change, [2-23](#)
external port registers, [A-45](#) to [A-63](#)
 AMI control (AMICTLx), [A-21](#), [A-47](#)
 control (EPCTL), [3-130](#)

F

FE, format extension, *See* SPORTs, word length
FFT accelerator, [6-3](#) to [6-20](#)
 block diagram, [6-6](#)
 chained DMA, [2-18](#)
 chain pointer register (FFTICP), [6-16](#)
 circular buffer addressing, [6-17](#)
 circular buffer chained DMA, [6-15](#)
 compute block, [6-5](#)
 data transfer types, [6-15](#)
 debug feature and strategy description, [6-20](#), [6-65](#)
 debugging, [6-27](#), [6-52](#)
 enable (ENABLE) bit, [6-7](#)
 horizontal, [6-13](#)
 H point coefficient buffer, [6-9](#)
 idle state, [6-7](#)
 interrupts, [6-18](#)
 interrupts, setting, [6-18](#)
 interrupt sources, DMA, [6-18](#)
 inverse, [6-15](#)
 large, [6-26](#)
 large, computation, [6-11](#)
 memory, coefficients, [6-6](#)
 memory, data, [6-6](#)
 memory, twiddle coefficients, [6-6](#)
 packed and unpacked date, [6-8](#)
 packing bits (FFT_CPACKIN, FFT_CPACKOUT), [6-14](#)
 processing state, [6-7](#)
 programming, [6-26](#)
 read state, [6-7](#)
 registers, [6-5](#)
 repeat (FFT_RPT) bit, [6-7](#)

(continued)

FFT accelerator
 repeat mode, [6-14](#)
 reset (FFT_RST) bit, [6-7](#)
 reset state, [6-7](#)
 small, computation, [6-11](#)
 special coefficient buffer, [6-10](#)
 special product, [6-12](#)
 start (START) bit, [6-7](#)
 storing small FFTs, [6-8](#)
 TCB structure, [6-18](#)
 vertical FFT example, [6-12](#)
 V point coefficient buffer, [6-9](#)
 write state, [6-7](#)

FFT bits
 interrupt (ACC_INT0, ACC_INT1), [6-18](#)

FIFO
 see also buffer
 data packing in IDP, [11-18](#)
 IDP, [11-6](#)
 IDP modes use, [11-11](#)
 receive, SPORT, [10-16](#)
 SPI, [15-9](#)
 SPI DMA, [15-22](#)
 to memory data transfer, [11-14](#)
 transmit, SPORT, [10-40](#)

FIG, frame ignore, *See* SPORTs, framed and unframed data

FIR
 channel control register (FIRCTL2), [A-82](#)
 control register 1 (FIRCTL1), [A-80](#)
 DMA status register (FIRDMASTAT), [A-85](#)
 MAC status register (FIRMACSTAT), [A-83](#)

FIR accelerator, [6-28](#) to [6-54](#)
 block diagram, [6-30](#)
 buffer, data, [6-33](#)
 chained DMA, [2-16](#)

- FIR accelerator *(continued)*
- coefficient memory, [6-32](#)
 - debug, [6-45](#)
 - decimation, [6-38](#)
 - delay line (TAP), [6-32](#)
 - DMA transfers, [6-42](#)
 - FIR_UPSAMP bit, [6-39](#)
 - formats, fixed-point, [6-42](#)
 - formats, floating-point, [6-42](#)
 - input sample, [6-33](#)
 - interpolation, [6-39](#)
 - interrupt sources, [6-45](#), [6-65](#)
 - MAC unit, [6-31](#), [6-32](#)
 - multiply accumulators, [6-32](#)
 - pre-fetch data, [6-33](#)
 - registers, [6-28](#), [6-32](#)
 - sample ratio (FIR_RATIO) bit, [6-39](#)
 - single rate operations, [6-38](#)
 - TAP delay line, [6-33](#)
 - window size (WINDOW) bit, [6-39](#)
- FIR bits
- channel auto iterate (FIR_CAI), [A-80](#)
 - channel complete interrupt
(FIR_CCINTR), [A-81](#)
 - channel number select (FIR_CH32–1),
[A-80](#)
 - DMA enable (FIR_DMAEN), [A-80](#)
 - rounding mode select (FIR_RND), [A-81](#)
 - tap length (TAPLEN), [A-82](#)
 - window size (WINDOW), [A-82](#)
- FIR filter inner loop, [3-97](#)
- flags
- flag interrupt mode (IRQxEN) bits, [A-5](#)
 - input/output (FLAGx) pins, [10-12](#), [15-9](#)
 - restriction with SPI bits, [7-4](#)
 - SPORT pins, [10-12](#)
- FLAGx pins, [10-12](#), [15-9](#)
- floating-point, [1-1](#)
- 40-bit, IIR, [6-61](#)
- FIR data format, [6-42](#)
- floating-point *(continued)*
- FIR multiplier, [6-31](#)
 - format select bit, IIR (IIR_FORTYBIT),
[A-89](#)
 - IIR accelerator, [6-59](#)
 - multipliers and adders, FFT, [6-5](#)
 - overflow, [6-18](#)
 - radix-2 complex FFT, [6-3](#)
 - rounding bit, FIR (FIR_RND), [A-81](#)
- force load mode register command, [3-52](#)
- framed versus unframed data, [10-26](#)
- frame sync
- A source (FSASOURCE) bit, [A-193](#)
 - early vs. late, [10-27](#)
 - frequencies, [10-8](#)
 - in multichannel mode, [10-32](#)
 - internal vs. external, [10-18](#)
 - output, synchronizing, [14-21](#)
 - PCG B source (FSBSOURCE) bit,
[14-13](#)
 - rates, setting, [10-29](#)
 - routing control (SRU_FS0) registers
(group C), [A-128](#)
 - signals, configuring, [10-11](#)
 - frame sync delay (MFD), [10-33](#)
 - frame sync required (FSR) bit, [A-155](#)
 - framing bits, [10-29](#), [10-31](#)
 - FRFS (frame on rising frame sync) bit,
[10-29](#), [10-31](#)
 - FS_BOTH (frame sync both) bit, [A-156](#)
- FSM, frame synchronization mode, *See*
SPORTs, framed and unframed data
- FSP, frame synchronization polarity, *See*
SPORTs, framed and unframed data
- FSR (frame sync required) bit, [A-155](#)
- full-duplex operation, specifications, [10-11](#)

Index

G

generators, optional reset, [23-42](#)
GM (get more data) bit, [A-233](#)
ground plane, in PCB design, [23-37](#)
group descriptions, signal routing unit, [9-17](#)

H

handshake, DMA, [2-30](#)
hardware reset, [23-3](#)
hold cycle (external bus) bit, [A-22](#), [A-49](#)
hold time
 inputs, [23-36](#)
 recognition of asynchronous input, [23-36](#)
hysteresis on $\overline{\text{RESET}}$ pin, [23-34](#)

I

I^2C port. *See* TWI controller
 I^2S
 (Tx/Rx on left channel first), [10-24](#)
ICLK (internal clock select) bit, [A-154](#), [A-159](#), [A-163](#)
idle cycle (external bus) bit, [A-23](#), [A-50](#)
IDP
 address, DMA, [11-20](#)
 address, ping-pong DMA, [11-21](#)
 buffer, [2-10](#)
 channel 0 diagram, [11-9](#)
 control (IDP_CTL0) register, [11-29](#), [A-175](#)
 control (IDP_CTL1) register, [11-28](#), [A-177](#), [A-178](#)
 (DAI) interrupt service routine
 steps, [11-23](#)
 DMA control registers, [A-174](#)
 DMA count (IDP_DMA_Cx) register, [11-20](#), [11-21](#)

IDP *(continued)*

 DMA index (IDP_DMA_Ix) register, [11-20](#), [11-21](#)
 DMA modify (IDP_DMA_Mx) register, [11-20](#), [11-21](#)
 FIFO (IDP_FIFO) register, [11-14](#), [11-15](#), [11-23](#)
 FIFO memory data transfer, [11-14](#)
 FIFO register (IDP_FIFO register), [11-14](#)
 illustrated, [11-2](#)
 interrupt driven transfers, [11-15](#)
 interrupts, [11-15](#), [11-27](#)
 memory data transfer, [11-14](#)
 packing modes, [11-10](#), [11-13](#)
 PDAP control (IDP_PDAP_CTL)
 register, [A-179](#)
 PDAP control (IDP_PP_CTL) register, [A-179](#)
 ping-pong DMA, [11-21](#)
 polarity of left-right encoding, [11-18](#)
 serial inputs, [11-6](#)
IDP bits
 bus hang disable (IDP_BHD), [11-25](#), [A-176](#)
 clear buffer overflow (IDP_CLROVR), [A-176](#)
 DMA enable (IDP_DMA_EN), [11-29](#), [A-176](#)
 DMA status (IDP_DMAt_STAT), [11-22](#), [A-183](#)
 enable (IDP_ENABLE), [11-29](#), [11-30](#), [A-176](#)
 FIFO number of samples
 (IDP_FIFOSZ), [A-183](#)
 FIFO samples exceed interrupt
 (IDP_FIFO_GTN_INT), [11-27](#)
 frame sync format (IDP_SMODEx), [11-15](#), [11-17](#), [11-27](#), [A-177](#)

- IDP bits *(continued)*
- IDP_DMA_EN bit
 - do not set, [11-16](#)
 - monitor number of samples
 - (IDP_NSET), [A-176](#)
 - PDAP clock edge
 - (IDP_PDAP_CLKEDGE), [11-27](#), [A-181](#)
 - PDAP enable (IDP_PDAP_EN), [11-29](#), [A-181](#)
 - PDAP input mask bits, [11-27](#)
 - PDAP packing mode
 - (IDP_PDAP_PACKING), [A-181](#)
 - PDAP reset (IDP_PDAP_RESET), [A-181](#)
 - ping-pong DMA enable (IDP_PING) bits, [A-178](#)
 - port select (IDP_PORT_SELECT), [A-180](#)
 - port select (IDP_PP_SELECT), [11-27](#)
 - reset (IDP_PDAP_RESET) bit, [A-181](#)
 - IDP_CTL0 (input data port control) register, [11-29](#), [A-175](#)
 - IDP_CTL1 (input data port control) register, [11-28](#), [A-177](#), [A-178](#)
 - IFS (internal frame sync select) bit, [A-155](#), [A-164](#)
 - IIR
 - global control register (IIRCTL1), [A-87](#)
 - IIR accelerator
 - 32- to 40-bit packing, [6-62](#)
 - chained DMA, [2-17](#)
 - chain pointer DMA, [6-62](#)
 - coefficient memory, [6-60](#)
 - control (IIRCTLx) register, [6-55](#)
 - data memory, [6-60](#)
 - debugging, [6-65](#)
 - DMA status (IIRDMASTAT) register, [6-56](#)
 - DMA transfers, [6-62](#)
 - IIR accelerator *(continued)*
 - floating-point operations, [6-59](#)
 - internal memory, [6-60](#)
 - MAC unit, [6-59](#)
 - operating modes, [6-61](#)
 - single step, [6-65](#)
 - throughput, [6-67](#)
 - transposed form 2 biquad, [6-58](#)
 - IIR accelerator registers
 - debug mode control (IIRDEBUGCTL), [6-55](#)
 - DMA status (IIRDMASTAT), [6-55](#)
 - global control (IIRCTLx), [6-55](#)
 - MAC status (IIRMACSTAT), [6-56](#)
 - IIR bits
 - 40-bit floating-point select (IIR_FORTYBIT), [A-89](#)
 - enable (IIR_EN), [A-87](#)
 - IISPx (serial port DMA internal index) registers, [2-4](#), [2-25](#)
 - IMSPI (serial peripheral interface address modify) register, [15-32](#)
 - IMSPx (SPORT DMA address modifier) registers, [2-5](#), [2-25](#)
 - index registers, [2-27](#)
 - INDIV (input divisor) bit, [A-8](#), [A-13](#)
 - input setup and hold time, [23-36](#)
 - input signal conditioning, [23-33](#)
 - input slave select enable (ISSEN) bit, [15-26](#), [A-233](#)
 - input synchronization delay, [23-30](#)
 - INTEN (DMA interrupt enable) bit, [15-26](#)
 - interconnections, master-slave, [15-3](#)
 - interleaving 16-bit addressing, DDR2, [3-65](#)
 - internal index, [2-27](#)
 - internal memory
 - DMA index (IDP_DMA_Ix) registers, [11-20](#), [11-21](#)
 - DMA index (IISPx) registers, [2-4](#), [2-25](#)

Index

- internal memory *(continued)*
- DMA modifier (IDP_DMA_Mx)
 - registers, [11-20](#), [11-21](#)
 - DMA modifier (IMSPx) registers, [2-5](#), [2-25](#)
- internal serial clock, setting, [10-29](#), [10-30](#)
- internal vs. external frame syncs, [10-18](#)
- interpolation, FIR filter, [6-39](#)
- interrupts
- chained, [2-46](#)
- interrupt
- enable interrupt on error (INTERR) bit, [15-26](#)
- interrupt and timer pins, [23-30](#)
- interrupt controller, DAI, [9-32](#)
- interrupts
- accelerator, [6-64](#)
 - channel priority, [2-48](#)
 - conditions for generating interrupts, [10-46](#)
 - data transfer, starting, [11-30](#)
 - digital applications interface, [9-33](#)
 - DMA slave, [4-19](#)
 - (enable RX status interrupt) bit, [A-247](#)
 - exception, [9-32](#)
 - external memory booting, [3-89](#)
 - FFT, [6-18](#)
 - FFT accelerator, [6-18](#)
 - FIFO to memory, [11-29](#)
 - latch (IRPTL) register, [15-26](#)
 - latch/mask (LIRPTL) registers, [15-26](#)
 - link ports, [4-16](#)
 - MAC status, FIR, [6-65](#)
 - masking, [2-45](#)
 - masking and latching, [4-20](#)
 - peripheral timers, [16-18](#)
 - polling, [2-45](#)
 - priority, [2-48](#)
- interrupts *(continued)*
- priority interrupt control registers (PICR), [B-4](#)
 - restrictions, [B-3](#)
 - SPORTs, [10-49](#)
 - SRC, [12-16](#)
 - status, fir, [6-45](#)
 - system, [9-32](#)
 - transfer completion, [2-46](#)
 - vector, sharing, [10-50](#)
- INVFSx (active low frame sync select for frame sync) bits, [14-13](#)
- I/O interface to peripheral devices, [10-1](#)
- I/O processor, [2-24](#)
- address bus (IOA), [2-25](#)
 - and addressing, [2-25](#)
 - buffer, [2-36](#)
 - DMA data, [2-32](#)
- bus priority, [2-42](#)
- bus priority, external port, [2-43](#), [A-20](#), [A-46](#)
- bus structure, [2-30](#)
- chain assignment, [2-33](#)
- chained DMA, [2-11](#)
- chain pointer (CPSPI) register, [2-11](#)
- chain pointer registers, [2-7](#)
- configuring DMA, [2-51](#)
- count registers, [2-6](#), [2-27](#)
- DMA channel registers, [2-36](#)
- IDP buffer, [2-10](#)
- miscellaneous external port parameter registers, [2-9](#)
- standard (non chained) DMA, [2-23](#)
- TCB memory allocation, [2-32](#)
- transfer types, [2-1](#), [2-2](#)
- ISSS (input service select) bit, [A-242](#)
- K**
- kernel boot timing, [23-23](#)

L

LAFS (late transmit frame sync select) bit, [10-24](#), [10-27](#), [A-155](#), [A-159](#)
 latchup, [23-33](#)
 latency
 input synchronization, [23-30](#)
 in SPORT registers, [10-55](#)
 link ports, [4-15](#)
 left-justified mode, [10-28](#), [C-5](#)
 SRC, [12-9](#), [A-186](#)
 SRC timing, [12-9](#)
 left-justified sample pair mode
 control bits, [10-29](#)
 Tx/Rx on FS rising edge, [10-24](#)
 length registers, DMA, [2-8](#)
 life counter, [18-1](#)
 Link buffer DMA chaining enable
 (LxCHEN) bit, [A-64](#)
 Link buffer DMA enable (LxDEN) bit, [A-64](#)
 Link buffer enable (LxEN) bit, [A-64](#)
 Link buffer transmit/receive (LxTRAN) bit, [A-64](#)
 link port, [4-5](#)
 boot bit settings, [23-22](#)
 booting, [23-21](#)
 DMA, [4-24](#)
 handshake timing, [4-6](#)
 initialization for boot, [23-22](#)
 interrupts, [4-16](#) to [4-21](#)
 token passing, [4-11](#)
 link port bits
 DMA channel count mask
 (DMACH_IRPT_MSK), [A-65](#)
 external transfer done mask
 (EXTTXFR_DONE_MSK), [A-65](#)
 receive request mask (LRRQ_MSK), [A-65](#)
 transmit request mask (LTRQ_MSK), [A-64](#)

Link port control (LCTL) register, [A-64](#)
 Link port receive request status (LxRRQ) bits, [A-66](#)
 link ports
 chained DMA, [2-15](#)
 transmit status (LTRQ) bit, [A-66](#)
 Link port transmit request status (LxTRQ) bits, [A-66](#)
 LIRPTL (interrupt) registers, [15-26](#)
 logical vs. physical address, [3-90](#)
 loopback mode
 timers, [16-19](#), [18-12](#), [19-7](#)
 loopback mode, SPI, [15-28](#)
 loop-back test, MLB, [8-16](#)
 low active transmit frame sync (LFS, LTFS and LTDV) bits, [A-196](#)
 LRFS (SPORT logic level) bit, [10-20](#)
 LSBF (least significant bit first) bit, [A-153](#)

M

MAC status (IIRMACSTAT) register, [6-56](#)
 making connections via the signal routing unit, [9-20](#)
 manual
 contents, [lxii](#)
 conventions, [lxx](#)
 new in this edition, [-xv](#)
 manual revisions, [lxv](#)
 mapping addresses, DDR2, [3-65](#)
 maskable interrupts, SPI, [15-26](#)
 mask bits, interrupt, [2-45](#)
 masking interrupts, link port, [4-20](#)
 master input slave output (MISO_x) pins, [15-8](#), [15-14](#)
 slave output, [15-14](#)
 master mode enable, [10-25](#), [10-30](#)
 master mode operation, SPI, [15-30](#)
 master out slave in (MOSI_x) pin, [15-8](#), [15-14](#)
 master-slave interconnections, [15-3](#)

Index

MCM, multichannel mode, *See* SPORTs
modes, multichannel mode
media local bus *See* MLB
memory
 coefficient, 6-33
 data transfer, FIFO, 11-14
 FFT data, 6-6
 IIR related, 6-60
 mapped IOP (RXSPI and TXSPI) buffer
 registers, 15-26
 memory-mapped registers, A-2
 TCB allocation for DMA, 2-32
 transfer types, 2-2
memory select (flags) programming
 (MSEN) bit, A-5
memory transfer types, 2-1
microcontroller, host, 23-7
MISCAx_I (signal routing unit external
 miscellaneous) register, 14-13
miscellaneous external port parameter
 registers, 2-9
miscellaneous signal routing
 (SRU_EXT_MISCx) registers (Group
 E), A-138
MISOx pins, 15-8, 15-14
MLB
 big-endian, 8-8
 buffer local channel, 8-8
 buffers, 8-9
 channel address, 8-6
 clock rate, 8-3
 configuring circular buffered DMA, 8-17
 configuring I/O mode using interrupts,
 8-16
 configuring ping-pong DMA, 8-17
 data structure, 8-7
 data transfer, core, 8-8
 DMA transfers, 8-11
 features, 8-3
 frame synchronization, 8-7

MLB (continued)
Generic Synchronous Packet Format
(GSPF), 8-7
interrupts, 8-14
I/O mode transfers, 8-9
I/O service requests, 8-9
little-endian, 8-8
logical channel, 8-6
loop-back testing, 8-14
MLBCLK pin, 8-3
MLBDAT pin, 8-3
MLBSIG pin, 8-3
MOST25 and MOST50, 8-2
multi-packet buffering, 8-12
PICR programming, 8-14
ping-pong DMA, 8-11
register descriptions, 8-4
registers, A-94 to A-113
RxStatus byte, 8-6
single-packet buffering, 8-12
specifications, 8-1
MLB bits
 buffer current address (BCA), 8-10,
 A-112
 bufferdepth (BD), A-113
 buffer final address (BFA), 8-10, A-112
 buffer ready, DMA (RDY), A-111
 buffer threshold (TH), 8-9, A-113
 channel enable (CE), A-107
 channel type select (CTYPE), A-107
 frame synchronization channel disable
 (FSCD), 8-7, A-105
 frame synchronization enable (FSE), 8-7,
 A-106
 frame synchronization physical channel
 count (FSPC), A-105
 little-endian mode (MLE), 8-8, A-95
 loop-back mode (LBM), 8-16
 MASK, 8-7
 next buffer end address (BEA), A-112

MLB bits *(continued)*

- next buffer start address (BEA), [A-112](#)

MLB registers

- base, DMA, [A-99](#)
- buffer, [8-5](#), [8-8](#), [A-111](#) to [A-113](#)
- channel control (MLB_CECRx), [8-7](#), [8-11](#), [8-13](#), [8-14](#), [8-16](#), [A-102](#)
- channel interrupt status (MLB_CICR), [8-14](#), [8-15](#), [A-99](#)
- channel status configuration (MLB_CSCRx), [8-7](#), [8-12](#), [8-13](#), [8-14](#)
- device control and configuration (MLB_DCCR), [8-4](#), [8-8](#), [8-16](#), [A-94](#)
- status, channel (MLB_CSCRx), [A-108](#)
- system data configuration (MLB_SDCR), [A-97](#)
- system mask configuration (MLB_SMCR), [8-15](#), [A-98](#)
- system status (MLB_SSCR), [8-4](#), [8-14](#), [8-15](#), [8-16](#), [A-96](#)

mode

- left-justified (IDP), [11-6](#)
- left-justified (SPORT), [10-28](#), [C-5](#)
- loopback (SPORT), [A-170](#)
- right-justified (IDP), [11-6](#)
- serial mode settings (IDP), [11-17](#)
- single channel double frequency (SPDIF), [13-12](#)
- standard serial, [10-25](#), [C-2](#)
- TDM (SPORT), [10-3](#)
- timer, [A-270](#)
- two channel (SPDIF), [13-12](#)

mode fault error (MME) bit, [15-26](#), [15-27](#)

mode fault (multimaster error) SPI DMA status (MME) bit, [15-26](#), [15-27](#)

modes, audio, [C-2](#) to [C-9](#)

MOSIx pins, [15-8](#), [15-14](#)

most significant byte first (MSBF) bit, [A-234](#)

MPEG-2 format, [13-17](#)

MSBF (most significant byte first) bit, [A-234](#)

MTxCCSx (serial port transmit compand) registers, [A-172](#)

MTxCCSy and MRxCCSy (multichannel compand select) registers, [10-12](#)

MTxCsX (serial port transmit select) registers, [A-171](#)

multibank operation with data packing, [3-36](#), [3-73](#)

multichannel compand select (MTxCCSy and MRxCCSy) registers, [10-12](#)

multichannel filtering, FIR, [6-49](#)

multichannel operation, [10-31](#)

multiple processor system example, [3-39](#), [3-76](#)

multiplexing

- clockout enable (CLKOUTEN), [A-11](#), [A-16](#)
- pins, [23-28](#) to [23-31](#)
- PWM pins, [A-6](#)
- $\overline{\text{RESETOUT}}$, [A-11](#), [A-16](#)

N

n greater than or equal to 512, repeat, [6-26](#)

normal frame sync, [10-27](#)

O

offset value, index registers, [2-27](#)

one shot frame sync A or B (STROBEx) bits, [14-13](#)

one shot option (STROBEB) bit, [14-13](#)

OPMODE (serial port operation mode) bit, [10-25](#), [10-29](#)

optimization

- core read, [3-42](#), [3-78](#)
- DDR2 reads, [3-76](#) to [3-79](#)
- SDRAM reads, [3-40](#)

- optimization *(continued)*
- SDRAM reads, example, [3-42](#)
 - SDRAM reads, external port DMA example, [3-43](#)
 - SDRAM throughput, [3-117](#)
- OR, logical, [11-32](#)
- P**
- package availability, [1-2](#)
- packing
- 16 to 32-bit packing (PACK) bit, [10-22](#), [10-45](#), [10-50](#), [A-154](#), [A-159](#), [A-163](#)
 - modes in IDP_PP_CTL, illustrated, [11-10](#)
 - serial peripheral interface (PACKEN) bit, [A-236](#)
- packing instructions, [3-90](#)
- page size (SDRAM), [A-55](#)
- parallel data acquisition port control (IDP_PP_CTL) register, [A-179](#)
- parallel data acquisition port (PDAP), [11-32](#)
- parameter registers, DMA, [2-36](#)
- PCG
- active low frame sync select for frame sync (INVFSx) bits, [14-13](#)
 - bypass mode, [14-13](#)
 - clock A source (CLKASOURCE) bit, [A-193](#)
 - clock input (CLKIN) pin, [17-5](#)
 - clock input source enable (CLKx_SOURCE_IOP) bit, [A-197](#)
 - clock with external frame sync enable (FSx_SYNC) bit, [A-197](#)
 - control (PCG_CTL_Ax) registers, [14-13](#), [A-191](#), [A-192](#)
 - division ratios, [14-18](#)
 - enable clock (ENCLKx) bit, [A-191](#), [A-192](#)
- PCG *(continued)*
- enable frame sync (ENFSx) bit, [A-191](#), [A-192](#)
 - frame sync A source (FSASOURCE) bit, [14-13](#), [A-193](#)
 - frame sync B source (FSBSOURCE) bit, [14-13](#), [A-193](#)
 - frame sync input source enable (CLKx_SOURCE_IOP) bit, [A-197](#)
 - frame sync with external frame sync enable (FSx_SYNC) bit, [A-197](#), [A-198](#)
 - one shot frame sync A or B (STROBEx) bits, [14-13](#)
 - one shot option, [14-13](#)
 - PCG_CTLA0 (control) register, [A-191](#), [A-192](#)
 - phase shift of frame sync, [14-16](#)
 - pulse width for frame sync (PWFSx) bit, [A-194](#)
 - pulse width (PCG_PW) register, [14-13](#)
 - synchronization with the external clock, [14-20](#)
- PCI (program control interrupt) bit, [2-13](#)
- PDAP
- enable (IDP_PDAP_EN) bit, [A-181](#)
 - port mask bits (IDP_Pxx_PDAPMASK), [A-180](#)
 - (rising or falling) clock edge (IDP_PDAP_CLKEDGE) bit, [A-181](#)
- PDAP control (IDP_PDAP_CTL) register, [A-179](#)
- peripheral devices, I/O interface to, [10-1](#)
- peripherals
- memory mapped, [3-10](#)
- peripheral timers
- external event watchdog (EXT_CLK) mode, [16-5](#), [16-15](#)
 - input/output (TMRx) pin, [16-5](#)
 - interrupts, [16-18](#)

- peripheral timers *(continued)*
- invalid conditions, [16-9](#)
 - modes, [16-5](#)
 - period, configuring, [16-4](#)
 - period equation, [16-11](#)
 - pulse width count and capture
 (WDTH_CAP) mode, [16-12](#)
 - pulse width modulation (PWMOUT)
 mode, [16-8](#)
 - rectangular signals, [16-10](#)
 - RTI instruction, [16-18](#)
 - single pulse generation, [16-11](#)
 - TIMERx pin, [16-6](#)
 - watchdog, [16-19](#)
 - word count (TMxCNT) registers, [16-6](#)
 - peripheral timers registers, [16-4](#)
 - high word period (TMxPRD) registers, [16-6](#)
 - high word pulse width (TMxW)
 registers, [16-6](#)
 - period (TMxPRD) registers, [16-4](#)
 - pulse width (TMxW) registers, [16-5](#)
 - timer control (TMxCTL), [16-6](#)
 - timer count (TMxCNT), [16-4](#), [16-6](#)
 - timer global status and control
 (TMSTAT), [16-4](#)
 - timer width (TMxW), [16-5](#)
 - timer word period (TMxPRD), [16-4](#)
 - word count (TMxCNT) registers, [16-4](#)
 - phase shift of frame sync, [14-16](#)
 - physical vs. logical address, [3-90](#)
 - PICR (programmable interrupt priority)
 registers, [B-4](#)
 - PICR register, [2-48](#), [B-6](#)
 - pin
 - buffer example, [9-7](#)
 - ping-pong DMA, [2-23](#), [2-24](#)
 - ping-pong DMA, IDP, [11-21](#)
 - pins
 - ACK, enabling, [A-22](#), [A-49](#)
 - descriptions, [22-18](#), [23-2](#)
 - external memory, [3-48](#)
 - FLAGx, [10-12](#)
 - pin states during SDRAM commands,
 [3-25](#)
 RESET, [23-34](#)
 - timer (through SRU), [16-5](#)
 - plane, ground, [23-37](#)
 - PLLBP (PLL bypass bit), [22-7](#), [A-11](#), [A-16](#)
 - PLLBP (PLL bypass mode) bit, [22-7](#), [22-15](#)
 - PLLDx (PLL divider) bits, [A-8](#), [A-10](#),
[A-11](#), [A-13](#), [A-16](#)
 - PLLM (PLL multiplier) bit, [A-8](#), [A-10](#),
[A-13](#), [A-16](#)
 - PLL start-up, [22-9](#)
 - PMCTL (power management control)
 register, [22-4](#), [A-7](#), [A-8](#), [A-10](#), [A-12](#),
[A-13](#), [A-16](#)
 - polarity
 - IDP left-right encoding, [11-18](#)
 - PWM double-update mode, [7-10](#)
 - PWM single update mode, [7-9](#)
 - SPDIF connections, [C-14](#)
 - SPI clock, [15-14](#), [15-29](#)
 - power management
 - additional information, [22-13](#)
 - bypass mode (PLL) example, [22-18](#)
 - clocking system, [22-2](#)
 - clock system, [22-2](#)
 - disabling peripherals, [22-10](#) to [22-12](#)
 - IDLE instruction use in, [22-13](#)
 - phase-locked loop (PLL), [22-2](#)
 - PLL, [22-9](#)
 - post divider example, [22-14](#)
 - registers (PMCTLx), [22-1](#)
 - VCO programming example, [22-15](#)
 - power management control (PMCTL)
 register, [A-7](#), [A-12](#)

Index

- power management control register
(PMCTL), [22-4](#), [A-8](#), [A-13](#)
- power management examples, [22-12](#)
- power management register, [A-10](#), [A-16](#)
- power savings, [22-6](#), [22-12](#)
- power savings tips, [22-13](#)
- power supply, monitor and reset generator, [23-42](#)
- power-up, SDRAM (SDPM) bit, [A-53](#)
- power-up, *See system design*
- power-up reset circuit, [23-42](#)
- preambles, S/PDIF, [C-16](#)
- precision clock generators. *See PCG*
- predictive address vs. real address, [3-14](#)
- predictive reads, disable bit (NO_OPT), [A-23](#), [A-50](#)
- printed circuit board design, [23-37](#)
- priority, interrupt, [2-48](#)
- processor, host, [23-7](#)
- processor core
- access to link buffers, [4-15](#)
- processor core, overview, [1-2](#)
- processor reset, [23-3](#)
- product details, [1-2](#)
- program control interrupt (PCI) bit, [2-13](#)
- program controlled interrupt bit (PCI),
[2-12](#), [2-13](#), [6-18](#), [10-48](#), [15-13](#),
[20-22](#), [A-167](#)
- programmable interrupt registers (PICRx), [B-4](#)
- programmable interrupts, listed, [2-48](#), [B-6](#)
- programming example, [6-71](#)
- programming model, [6-68](#)
- TWI controller, [21-19](#)
- pulse, clock, in serial ports, [10-11](#)
- pulse, frame sync delay in serial ports, [10-33](#)
- pulse, frame sync formula, [10-9](#)
- pulse width count and capture
(WDTH_CAP) mode, [16-12](#)
- pulse width modulation (PWMOUT)
mode, [16-8](#)
- PWM
- 16-bit read/write duty cycle registers, [7-7](#)
 - accuracy in, [7-23](#)
 - block diagram, [7-2](#)
 - center-aligned paired PWM
 - double-update mode, [7-10](#)
 - channel duty control (PWMA, PWMB)
registers, [A-73](#)
 - crossover mode, [7-14](#)
 - equations, [7-8](#) to [7-12](#)
 - global control (PWMGCTL) register,
[A-67](#)
 - global status (PWMGSTAT) register,
[A-69](#)
 - switching frequency equation, [7-6](#)
- PWM bits
- crossover (PWM_AXOV,
PWM_BXOV), [7-14](#)
- PWMGCTL (pulse width modulation
global control) register, [A-67](#)
- PWMGSTAT (pulse width modulation
global status) register, [A-69](#)
- PWMOUT (pulse width modulation)
mode, [16-8](#)
- ## R
- real time clock
- see RTC*
- real-time clock. *See RTC*
- receive busy (overflow error) SPI DMA
status (SPIOVF) bit, [A-238](#)
- receive data, serial port (RXSPx) registers,
[2-10](#)
- receive data (RXSPI) buffer, [15-8](#)
- receive overflow error (SPIOVF) bit,
[15-26](#), [15-37](#), [15-38](#)
- receive shift (RXSR) register, [15-8](#)
- refresh rate control (SDRRC) register, [A-58](#)

- refresh rate in SDRAM, 3-33
 register drawings, reading, A-2
 registers, *See* peripheral specific registers
 register writes and effect latency, SPORTs, 10-40
 reset
 generators, 23-41
~~RESET~~ after power-up, 23-3
~~RESET~~ pin, 23-34
 input hysteresis, 23-34
 resolution (PWM), 7-23
 restrictions
 external memory access, 3-132
 interrupts, B-3
 SDRAM, 3-27
 SPORTs, 10-32
 right justified mode, C-5
 right-justified mode
 SRC, 12-9
 SRC, timing, 12-9
 rotating DMA priority, 2-44
 ROVF_A or TUVF_A (channel A error status) bit, A-158, A-161
 ROVF_A or TUVF_A (serial port error status) bits, A-158, A-161
 ROVF_B or TUVF_B (channel B error status) bit, A-157, A-161, A-166
 RS-232 device, restrictions, 10-12
 RTC
 alarm, 18-7
 alarm clock features, 18-7
 block diagram, 18-8
 calibration, 18-2, 18-8
 clocking, 18-3
 clock register (RTC_CLOCK), 18-3
 clock requirements, 18-3
 counters, 18-4
 digital watch, 18-5
 digital watch features, 18-1
 disabling, 18-3
 RTC *(continued)*
 error, 18-8
 event flags, 18-14, 18-15
 flags (list), 18-15
 initialization register (RTC_INIT), 18-3
 pin descriptions, 18-3
 pins, 18-3
 power-up, 18-4
 programming, 18-12
 registers, 18-3
 reset, 18-4
 specifications, 18-1
 status register (RTC_STAT), 18-3
 stopwatch, 18-8
 stopwatch function, 18-8
 RTC_ALARM (RTC alarm) register, 18-7
 RTC alarm (RTC_ALARM) register, 18-7
 RTC_STAT (RTC status) register, 18-14
 RTC status (RTC_STAT) register, 18-14
 RTC stopwatch count (RTC_SWCNT) register, 18-8
 RTC_SWCNT (RTC stopwatch count) register, 18-8
 running reset, 23-4
 RXFLSH (flush receive buffer) bit, 15-35, 15-36, 15-38
 RXS_A (data buffer channel B status) bit, A-158, A-161, A-166
 RXSPI, RXSPIB (SPI receive buffer) registers, 15-20, 15-26
 RXSPx (serial port receive buffer) registers, 2-10
 RXSR (SPI receive shift) register, 15-8
 RXS (SPI data buffer status) bit, A-241
 RX_UACEN (DMA receive buffer enable) bit, 20-5

Index

S

sample/window based processing mode,
 IIR, [6-38](#), [6-61](#)
sampling clock period, UART, [20-12](#)
sampling point, UART, [20-11](#)
saving power, [22-6](#), [22-12](#)
SCHEN_A and SCHEN_B (serial port
 chaining enable) bit, [A-156](#), [A-160](#),
 [A-164](#)
SDEN (serial port DMA enable) bit,
 [A-156](#), [A-160](#), [A-164](#)
SDRAM, [3-36](#), [3-73](#)
 bus errors, [3-44](#), [3-80](#)
 clock equation, [3-33](#)
 core address mapping, [3-27](#)
 multibank operation, [3-35](#), [3-71](#)
 refresh rate, [3-33](#)
 refresh rate control register (SDRRC),
 [A-58](#)
 registers, [A-51](#) to [A-59](#)
 restrictions, [3-27](#)
 status register (SDSTAT), [A-55](#)
SDRAM bits
 CAS latency (SDCL), [A-53](#)
 column address width (SDCAW), [A-54](#)
 disable clock and control (DSDCTL),
 [A-53](#)
 external data path width (X16DE), [A-54](#)
 force auto refresh (Force AR), [A-54](#)
 force load mode register write (Force
 LMR), [A-55](#)
 force precharge (Force PC), [A-54](#)
 optional refresh (SDORF), [A-54](#)
 page size is 128 words (PGSZ 128), [A-55](#)
 pipeline option with external register
 buffer (SDBUF), [A-55](#)
 power-up mode (SDPM), [A-53](#)
 power-up sequence start (SDPSS), [A-54](#)
 RAS setting (SDTRAS), [A-53](#)
 RDC setting (SDTRCD), [A-55](#)

refresh delay (RDIV), [A-59](#)
row address width (SDRAW), [A-55](#)
RP setting (SDTRP), [A-53](#)
self-refresh enable (SDSRF), [A-54](#)
WR setting (SDTWR), [A-54](#)
SDRAM controller, [3-17](#)
 addressing (16-bit), [3-28](#) to [3-30](#)
 calculating refresh rate, [3-33](#)
 clock frequencies, [3-6](#)
 external memory access timing, [3-5](#)
 power-up sequence, [A-54](#), [A-55](#)
 read/write command, [3-22](#)
 refresh rate (SDRRC) register, [3-32](#)
 setting bank column address width, [A-55](#)
 write before precharge (SDTWR) bit,
 [3-5](#)
SDRAM controller commands
 auto-refresh, [3-24](#)
 bank activate, [3-21](#)
 command pin states, [3-25](#)
 load mode register, [3-21](#)
 NOP/command inhibit, [3-24](#)
 precharge all, [3-22](#)
 self-refresh, [3-44](#), [3-79](#)
 single precharge, [3-21](#)
SENDZ (send zeros) bit, [A-233](#)
serial clock (SPORTx_CLK) pins, [10-11](#)
serial communications, [20-7](#)
serial inputs, [11-6](#)
serial peripheral interface, *See* SPI
setting up DMA on SPORT channels,
 [10-46](#)
setup time, inputs, [23-36](#)
shared memory
 see also external port
shift register, [2-27](#)
short word data, I/O processor, [2-27](#)
signal routing unit external miscellaneous
 (MISCAx) registers, [14-13](#)
signal routing unit *See* SRU, DAI

signal routing unit (SRU), [16-6](#)
 signals
 PWM waveform generation and, [16-10](#)
 sensitivity in serial ports, [10-6](#)
 serial port, [10-8](#), [10-11](#)
 SPORT, [10-5](#)
 timer, [16-5](#)
 single channel double frequency mode, [13-12](#)
 single processor system example, [3-39](#), [3-76](#)
 single rate operations FIR, [6-38](#)
 software reset, [23-4](#)
 SP1PDN (SPORT1 clock enable) bit, [A-11](#), [A-16](#)
 SP3PDN (SPORT3 clock enable) bit, [A-12](#), [A-17](#)
 SPCTLx control bit comparison in four SPORT operation modes, [10-22](#)
 SPCTLx control bits for left-justify mode, [10-25](#)
 SPCTLx (serial port control) registers, [10-11](#), [10-12](#), [10-21](#)

S/PDIF

See also S/PDIF bits; S/PDIF registers

AAC compressed format, [13-17](#)
 AC-3 format, [13-17](#)
 audio standards, [13-13](#)
 biphase encoding, [13-5](#)
 block structure, [C-11](#)
 clock (SCLK) input, [13-5](#)
 compressed audio data, [13-16](#)
 DTS format, [13-17](#)
 frame sync (LRCLK) input, [13-5](#)
 MPEG-2 format, [13-17](#)
 non-linear audio data, [13-16](#)
 output routing, [13-8](#)
 pin descriptions, receiver, [13-4](#)
 pin descriptions, transmitter, [11-3](#), [13-3](#)
 preambles, [C-16](#)
 programming guidelines, [13-6](#)

(*continued*)

S/PDIF

 serial clock input, [13-6](#)
 serial data (SDATA) input, [13-5](#)
 single-channel, double-frequency format, [13-12](#)
 subframe format, [C-13](#)
 timing, [13-4](#)
 two channel mode, [13-12](#)

S/PDIF bits

 biphase error (DIR_BIPHASEERROR), [A-208](#)
 channel status buffer enable (DIT_CHANBUF), [A-201](#)
 channel status byte 0 A (DIT_B0CHANL), [A-201](#)
 channel status byte 0 B (DIT_B0CHANR), [A-201](#)
 channel status byte 0 for subframe A (DIR_B0CHANL), [A-208](#)
 channel status byte 0 for subframe B (DIR_B0CHANR), [A-208](#)
 disable PLL (DIR_PLLDIS), [A-206](#)
 frequency multiplier (DIT_FREQ), [A-200](#)
 lock error (DIR_LOCK), [A-205](#)
 lock receiver status (DIR_LOCK), [A-208](#)
 mute receiver (DIR_MUTE), [A-206](#)
 mute transmitter (DIT_MUTE), [A-200](#)
 non-audio frame mode channel 1 and 2 (DIR_NOAUDIOLR), [A-207](#)
 non-audio subframe mode channel 1 (DIR_NOAUDIOL), [A-207](#)
 parity biphase error (DIR_BIPHASE), [A-205](#)
 parity (DIR_PARITYERROR), [A-208](#)
 select single channel double frequency mode channel (DIT_SCDF_LR), [A-200](#)

Index

S/PDIF bits (*continued*)
 serial data input format
 (DIT_SMODEIN), [A-201](#)
 single channel double frequency channel
 select (DIR_SCDF_LR), [A-205](#)
 stream disconnected
 (DIR_NOSTREAM), [A-208](#)
 transmit single channel double frequency
 enable (DIT_SCDF), [A-200](#), [A-205](#)
 transmitter enable (DIT_EN), [A-200](#)
 validity bit A (DIT_VALIDL), [A-201](#)
 validity bit B (DIT_VALIDR), [A-201](#)
 validity (DIR_VALID), [A-207](#)

S/PDIF registers
 channel A transmit status
 (SPDIF_TX_CHSTA), [A-202](#), [A-203](#)
 channel B transmit status
 (SPDIF_TX_CHSTB), [A-203](#), [A-204](#)
 left channel status for sub-frame A
 (DIRCHANL), [A-209](#)
 receiver status (DIRSTAT), [A-206](#)
 SRU control, [13-5](#), [13-8](#)
 transmit control (DITCTL), [13-6](#),
 [A-199](#)

SPDIF_TX_CHSTA (Sony/Philips digital interface channel status) register,
 [A-202](#), [A-203](#)

special IDP registers, [A-148](#)

SPEN_A (serial port channel A enable) bit,
 [A-153](#), [A-159](#)

SPI
See also SPI bits; SPI registers
AD1855 DAC and, [15-10](#)
address, TCB, [15-33](#)
block diagram, [15-8](#)
booting, [23-12](#), [23-15](#)
boot packing, [23-17](#)
chained DMA, [2-14](#)
chaining, DMA, [15-13](#), [15-21](#), [15-23](#),
 [15-25](#), [15-32](#)

SPI (*continued*)
 change clock polarity, [15-29](#)
 changing configuration, [15-29](#)
 clock phase, [15-14](#)
 clock (SPICLK) pin, [15-14](#)
 clock (SPICLK) signal, [15-8](#)
 configuring and enabling, [15-31](#), [15-33](#)
 core transfers, [15-31](#), [15-32](#)
 DMA, switching from transmit to receive mode, [15-34](#)
 examples, timing, [15-17](#)
 examples, transfer protocol, [15-15](#)
 features, [15-2](#)
 functional description, [15-8](#)
 interconnections, master-slave, [15-3](#)
 interface signals, [15-3](#)
 interrupt, [15-24](#)
 loopback mode, [15-28](#)
 master boot mode, [23-12](#)
 master input slave output (MISO_x) pins, [15-8](#)
 master mode, [15-30](#)
 master mode operation, configuring for, [15-30](#)
 master out slave in (MOSI_x) pins, [15-8](#)
 master-slave interconnections, [15-3](#)
 operation, master mode, [15-31](#), [15-33](#)
 operations, [15-30](#)
 polarity, clock, [15-14](#), [15-29](#)
 receive data (RXSPI) buffer, [15-8](#), [15-30](#)
 registers, [A-232](#)
 slave boot mode, [23-15](#)
 SPI_{D/S} pin, [15-32](#), [A-233](#)
 switching from receive to transmit mode, [15-34](#), [15-35](#)
 system, configuring and enabling bits, [A-232](#)
 transfer formats, [15-14](#)
 transmit data (TXSPI) buffer, [15-8](#)

- | | |
|---|-------------|
| SPI | (continued) |
| transmit underrun error (SPIUNF) bit,
15-26, 15-37, 15-38, A-238 | |
| TXFLSH (flush transmit buffer) bit,
15-35, A-236 | |
| SPI bits | |
| chained DMA enable (SPICHEN_A and
SPICHEN_B), A-238 | |
| chain loading status (SPICHS), A-239 | |
| clock phase (CPHASE), A-234 | |
| clock polarity (CLKPL), A-234 | |
| device select enable (DSxEN), 15-31,
15-33 | |
| DMA interrupt enable (INTEN), 15-26 | |
| enable interrupt on error (INTERR),
15-26 | |
| enable (SPIEN), A-235 | |
| FIFO clear (FIFOFLSH), A-238 | |
| flush receive buffer (RXFLSH), 15-35,
15-36, A-236 | |
| flush transmit buffer (TXFLSH), A-236 | |
| get more data (GM), A-233 | |
| input slave select enable (ISSEN), 15-26 | |
| input slave select (ISSEN), A-233 | |
| internal loop back (ILPBK), A-236 | |
| low priority interrupt (SPILI), 15-26 | |
| master select (SPIMS), A-234 | |
| MISO disable (DMISO), A-233 | |
| mode fault error (MME), 15-26, 15-27 | |
| most significant byte first (MSBF),
A-234 | |
| open drain output select (OPD), A-235 | |
| packing enable (PACKEN), A-236 | |
| program controlled interrupt bit (PCI),
15-13 | |
| receive overflow error (SPIOVF), 15-26,
15-37, 15-38 | |
| seamless transfer (SMSL), A-236 | |
| send zero (SENDZ), A-233 | |
| sign extend (SGN), A-236 | |
| SPI bits | (continued) |
| word length (WL), A-234 | |
| SPICHEN_A and SPICHEN_B (SPI
DMA chaining enable) bits, A-156,
A-160, A-164 | |
| SPICLK (SPI clock) pins, 15-14 | |
| SPICLK (SPI clock) signal, 15-8 | |
| SPICTL (SPI port control) registers, A-233 | |
| SPIDMAC (SPI DMA control) register,
15-21, A-237 | |
| <u>SPIDS</u> (SPI device select) pin, 15-14 | |
| SPIDS status, <i>See</i> ISSS bit | |
| SPI general operations, 15-4, 15-5 | |
| SPILI (SPI low priority interrupt) bit,
15-26 | |
| SPI master mode operation, 15-30 | |
| SPIOVF (SPI receive overflow error) bit,
15-26, 15-37, 15-38 | |
| SPIPDN (SPI clock enable) bit, A-11,
A-12, A-17 | |
| SPI registers | |
| DMA configuration (SPIDMAC),
15-21, 15-32, 15-34, A-237 | |
| flag (SPIFLGx), A-242 | |
| interrupt latch (IRPTL), 15-26 | |
| interrupt latch/mask (LIRPTL), 15-26 | |
| interrupt (LIRPTL), 15-26 | |
| receive buffer (RXSPI), 2-10 | |
| receive control (SPICTL, SPICTLB),
A-232 | |
| RXSR (SPI receive shift), 15-8 | |
| SPIBAUD (baud rate) register, A-239 | |
| status (SPISTAT), 15-26, 15-37, A-240 | |
| status (SPISTAT, SPISTATB), A-240 | |
| transmit buffer (TXSPI), 15-30, A-239 | |
| TXSR (SPI transmit shift), 15-8 | |
| SPISTAT, SPISTATB (SPI status
registers, A-240 | |
| SPIUNF (SPI transmit underrun error) bit,
15-26, 15-37, 15-38 | |

Index

- SPORT bits, [10-33](#)
 B channels enable (MCEB), [A-171](#)
 chained DMA enable (SCHEN), [10-46](#),
 [10-47](#), [10-57](#)
 chained DMA enable (SPICHEN),
 [A-156](#), [A-160](#), [A-164](#)
 channel A enable (SPEN_A), [A-153](#),
 [A-159](#)
 channel error status (ROVF_A or
 TUVF_A), [10-16](#), [A-158](#), [A-161](#)
 clock, internal clock (ICLK), MSTR (I²S
 mode only), [A-154](#)
 clock, MSTR (I²S mode only), [A-159](#),
 [A-163](#)
 clock rising edge select (CKRE), [10-8](#)
 control bit comparison, [10-22](#)
 current channel selected (CHNL), [A-171](#)
 data independent transmit/receive frame
 sync (DIFS), [A-155](#), [A-159](#)
 data type (DTYPE), [10-22](#)
 DMA chaining status (DMACHSxy),
 [A-171](#)
 DMA enable (SDEN), [A-156](#), [A-160](#),
 [A-164](#)
 DMA status (DMASxy), [A-171](#)
 DXS_B (data buffer status), [A-157](#),
 [A-161](#), [A-166](#)
 frame on rising frame sync (FRFS),
 [10-29](#), [10-31](#)
 FS both enable (FS_BOTH), [A-156](#)
 internal frame sync select (IFS), [A-155](#),
 [A-164](#)
 internal serial clock (ICLK), [10-8](#)
 late frame sync (LAFS), [A-155](#), [A-159](#)
 loopback mode (SPL), [A-170](#)
 multichannel frame delay (MFD), [A-170](#)
 multichannel mode enable (MCEA),
 [A-170](#)
 number of channels (NCH), [10-35](#)
- SPORT bits *(continued)*
 number of multichannel slots (NCH),
 [A-170](#)
 operation mode (OPMODE), [10-19](#),
 [10-22](#), [10-23](#), [10-25](#), [10-29](#), [10-31](#)
 program controlled interrupt bit (PCI),
 [10-48](#), [A-167](#)
 receive underflow status (ROVF_A or
 TUVF_A), [A-158](#), [A-161](#)
 serial word length (SLEN), [10-25](#), [10-29](#),
 [10-31](#)
- SPORT modes
(I²S), [10-21](#)
I²S (Tx/Rx on left channel first), [10-24](#)
left-justified, [10-24](#), [10-28](#), [C-5](#)
loopback, [10-54](#)
multichannel, [10-31](#)
standard DSP, [10-24](#), [10-25](#), [C-2](#)
- SPORT registers
 channel selection, [10-36](#)
 control, [10-22](#)
 control (SPCTLx), [10-11](#), [10-12](#), [10-21](#),
 [10-22](#)
 divisor (DIVx), [10-11](#)
 multichannel control (SPMCTLxy),
 [10-23](#), [10-33](#)
 receive buffer (RXSPx), [10-14](#), [10-15](#),
 [10-40](#), [10-41](#), [C-4](#)
 SPCTLx (serial port control), [A-151](#)
 transmit buffer (TXSPx), [10-14](#), [10-15](#),
 [10-40](#), [C-4](#)
 transmit compand (MTxCsX,
 MTxCCSx), [A-172](#)
- SPORTs, [10-60](#)
 See also SPORT bits, modes, registers
 128-channel TDM, [10-3](#)
 address, DMA, [10-46](#)
 address, TCB and, [10-56](#)
 buffer hang disable (BHD) bit, [10-54](#)
 buffers, data, [10-40](#)

SPORTs (*continued*)

- chained DMA, [2-14](#)
- chain insertion (DMA), [2-36](#)
- chain insertion mode (DMA), [10-48](#)
- channel number (quantity) select (NCH bit), [10-35](#)
- clock divisor equation, [10-9](#)
- clock frequency equation, [10-9](#)
- clock (SCLKx) pins, [10-11](#)
- clock signal options, [10-8](#)
- companding and data type bit (DTYPE), [10-14](#)
- companding (compressing/expanding), [10-3](#)
- configuring frame sync signals, [10-11](#)
- configuring standard DSP serial mode, [10-25](#)
- control bit comparison, [10-22](#)
- data type, sign-extend, [10-12](#)
- data type, zero-fill, [10-12](#)
- data type bit (DTYPE), [10-22](#)
- debugging, [A-170](#)
- divisor (DIVx) register, [10-8](#), [10-18](#), [10-26](#), [10-29](#)
- DMA chaining, [10-47](#)
- DMA channels, [10-45](#)
- duplex, full, [10-11](#)
- enabling B channels, [A-171](#)
- equation
 - frame sync frequency, [10-9](#)
- examples, normal vs. alternate framing, [10-28](#)
- features, [10-2](#)
- finding currently selected channel, [A-171](#)
- flag pins, [10-12](#)
- FLAGx pins, [10-12](#)
- framed and unframed data, [10-26](#)
- framed vs. unframed data example, [10-28](#)
- frame sync and serial word length, [10-10](#)

SPORTs (*continued*)

- frame sync delay, [10-33](#)
- full-duplex operation, [10-11](#)
- input/output (FLAGx) pins, [10-12](#)
- internal clock selection, [10-8](#)
- interrupts, [10-46](#), [10-49](#), [10-50](#)
- I/O processor bus and, [10-47](#)
- latency in writes, [10-55](#)
- left-justified mode control bits, [10-29](#)
- loopback mode, [10-54](#)
- masking interrupts, [10-50](#)
- master mode enable, [10-30](#)
- operation modes, changing, [10-21](#)
- operation modes, listed, [10-21](#)
- operation modes, standard DSP serial, [10-25](#), [C-2](#)
- packing enable (PACK) bit, [10-22](#), [10-45](#), [10-50](#)
- pairing, [10-32](#), [10-34](#)
- receive buffers, [10-41](#)
- serial clock pins, [10-11](#)
- serial word length and frame sync, [10-10](#)
- setting frame sync rates, [10-29](#)
- signal sensitivity, [10-6](#)
- SPORTx_DA and SPORTx_DB
 - channel data signal, [10-11](#)
- SPORTx_FS (serial port frame sync) pins, [10-11](#)
- transmit buffers, [10-40](#)
- transmit underflow status (TUVF_A) bit, [A-158](#), [A-161](#)
- transmit valid data signal (SPORTx_TDV_0), [10-33](#)
- Tx/Rx on FS rising edge, [10-24](#)
- using with SRU, [10-5](#)
- warnings and cautions, [10-42](#)
- word length, [10-10](#)
- SPTTRAN (serial port data direction control) bit, [A-157](#), [A-160](#), [A-165](#)

Index

- SRC
block diagram, 12-6
clocking, 12-11, C-7
control (SRCCTLx) register, A-185
frame sync signal, 12-4
interrupts, 12-16
mute (SRCMUTE) register, A-189
ratio (SRCRAT) register, A-189
right justified mode, 12-14
right-justified mode, 12-9
right-justified mode, timing, 12-9
time division multiplexing mode, 12-11, 12-14, C-8
- SRC bits
auto mute (SRC0_AUTO_MUTE), A-185
bypass (SRC0_BYPASS), A-186
de-emphasis (SRC0_DEEMPHASIS), A-186
dither select (SRC0_DITHER), A-186
enable (SRC0_ENABLE), A-187
hard mute (SRC0_HARD_MUTE), A-185
matched phase select (SRC0_MPASE), A-187, A-189
serial input format (SRC0_SMODEIN), A-186
serial output format (SRC0_SMODEOUT), A-186
soft mute (SRC0_SOFTMUTE), A-186
word length, output (SRC0_LENOUT), A-187
- SRU
bidirectional pin buffer, 9-14
buffers, 9-14
connecting peripherals with, 9-16
connecting through, 9-20
frame sync routing control (SRU_FSx) registers, A-128
group A (clock) signals, 9-24, A-218
- SRU *(continued)*
group E (miscellaneous) signals, A-138 to A-140
inputs, 9-16
outputs, 9-16
register use of, 9-20
serial ports and, 10-5
signal groups, 9-18, 9-18 to 9-19
signal groups, defined, 9-16, 9-17
signal sources, clock, A-118
signal sources, frame sync, A-128
signal sources, miscellaneous, A-138
signal sources, pin signal, A-132
SPORT signal connections, 10-5
- SRU2
group B (pin assignment) signals, A-223
group C (pin enable) signals, A-226
- SRU registers
clock (SRU_CLKx), A-118
frame sync (SRU_FSx), A-128
miscellaneous (SRU_EXT_MISCx), A-138
pin assignment (SRU_PINx) registers (group D), A-132
pin enable (SRU_PINENx) registers, A-141, A-145
pin signal (SRU_PINx), A-132
SRU_DATx (SRU data) registers, A-123
SRU_EXT_MISCx (SRU external miscellaneous) registers, A-138
SRU_FSx (SRU frame sync routing control) registers, A-128
SRU_PINENx (SRU pin buffer enable) registers, A-141, A-145
SRU_PINGx_STAT (ping-pong DMA status) register, A-182, A-183
SRU_PINx (pin signal assignment) registers, A-132
standard DSP serial mode, 10-25, C-2

- starting an interrupt driven transfer, 11-29, 11-30
- status registers, DMA, 2-36
- stopwatch function, RTC, 18-8
- STROBEA (one shot frame sync A) bit, 14-13, A-195
- STROBEB (one shot frame sync B) bit, 14-13, A-195
- strobe period, 14-13
- supervisory circuits, 23-41
- support, technical or customer, -lxvi
- switching from receive to transmit DMA, 15-35
- synchronization with the external clock, 14-20
- synchronizing frame sync output, 14-21
- SYSCTL register
- external port data pin mode select (EPDATA) bits, A-6
 - interrupt request enable (IRQxEN) bits, A-5
 - memory select (MSEN) bit, A-5
 - pulse width modulation select (PWMx) bits, A-6
 - rotating priority bus arbitration (DCPR) bit, 2-42
 - timer (flag) expired mode (TMREXPEN) bit, A-5
- SYSCTL (system control) register, A-4
- system, 23-33
- system control register. *See* SYSCTL register
- system design
- baud rate, init value, 23-13
 - boot times, kernel, 23-22
 - bypass capacitors, 23-36
 - clock distribution, 23-33
 - conditioning input signals, 23-33
 - crosstalk, 23-37
 - decoupling capacitors, 23-36
 - system design (continued)
 - designing for high frequency operation, 23-33
 - generators, reset, 23-42
 - ground plane, 23-37
 - hold time, inputs, 23-36
 - input setup and hold time, 23-36
 - input signal conditioning, 23-33
 - kernel, boot, 23-22
 - latchup, 23-33
 - latency, input synchronization, 23-30
 - link port booting, 23-21
 - pin descriptions, 23-2
 - plane, ground, 23-37
 - PLL start-up, 22-3
 - power options, 22-6, 22-12
 - power supply, monitor and reset generator, 23-42
 - power-up, 22-3
 - recommendations and suggestions, 23-37
 - RESET pin, 23-34
 - resetting the processor, 23-3
 - VCO encodings, 22-5
 - system interrupt, 9-32
- T
- TCB, 2-13 to 2-21, 2-32 to 2-35
- TCB chain loading, 2-11
- technical or customer support, -lxvi
- technical support, -lxvi
- test
- loopback, SPI, A-236
 - test mode
 - DAI use in, 9-10, 9-40
 - data buffer use in, 2-11
 - loopback, SPI, 15-28, A-236
 - SPI, 4-21, 15-27
 - SPORT, 10-53, 10-60, 16-19, 18-11, 19-7, 21-18

Index

- system, 23-33
- throughput
 - IIR, 6-67
- THR register empty (THRE) flag, 20-10, 20-12
- time division multiplexed (TDM) mode, 10-31, 12-11, C-7
- timer, configuring, A-270
- timer registers, A-269
 - timer control (TMxCTL), A-270
 - timer status (TMxSTAT), A-271
- timers, UART, 20-6
- timer *See* peripheral timers, core timer
- timing
 - external memory accesses, 3-5
 - kernel boot, 23-23
 - link port handshake, 4-6
 - PWM, 7-8, 7-9
 - S/PDIF, 13-4
 - SPI clock, 15-17
 - SPI slave, 15-17
 - SPI transfer protocol, 15-15
 - SPORT bits, 10-25
 - SPORT framed vs. unframed data, 10-28
 - SPORT normal vs. alternate framing, 10-28
 - SRC, 12-9
- TIMOD (transfer initiation mode) bit, 15-26, 15-30
- TMRPDN (timer clock enable) bit, A-11, A-12, A-17
- TMSTAT (peripheral timer global status and control) register, 16-4
- TMxCNT (peripheral timer word count) registers, 16-4, 16-6
- TMxCTL (peripheral timer control) registers, 16-6
- TMxCTL (timer control) registers, A-270
- TMxPRD (peripheral timer period) registers, 16-4
- TMxSTAT (timer global status and control) register, A-271
- TMxW (peripheral timer width) registers, 16-5
- TMxW (peripheral timer word pulse width) registers, 16-6
- token passing, link ports, 4-11
- T_PRDHx (timer period) registers, 16-4, 16-6
- transfer, data, 2-46
- transfer control block, *See* DMA TCB
- transfer control block *See* DMA TCB
 - transfer direction, external port, 2-24
 - transfer initiation and interrupt (TIMOD) mode, 15-26
 - transmit and receive data buffers (TXSPxA/B, RXSPxA/B), 10-40
 - transmit and receive SPORT data buffers (TXSPxA/B, RXSPxA/B), 10-40
 - transmit data (TXSPI) buffer, 15-8
 - transmit shift (TXSR) register, 15-8
 - TUVF_A (channel error status) bit, A-158, A-161
- TWI controller
 - architecture, 21-6
 - block diagram, 21-7
 - bus arbitration, 21-9
 - call address, 21-14
 - clocking, 21-8
 - error, 21-11, 21-20, 21-24
 - fast mode, setting, 21-15
 - programming model, 21-19
 - start and stop conditions, 21-11
 - transferring data, 21-8
- TWI controller bits
 - address not acknowledged (TWIANAK), A-261
- buffer write error (TWIWERR), A-262

TWI controller bits (*continued*)
 clock high (TWICLKHI), [A-116](#),
[A-117](#), [A-254](#)
 clock low (TWICLKLOW), [A-115](#),
[A-116](#), [A-254](#)
 data not acknowledged (TWIDNAK),
[A-261](#)
 data transfer count (TWIDCNT), [A-259](#)
 enable (TWIEN), [A-254](#)
 fast mode (TWIFAST), [A-258](#)
 general call enable (TWIGCE), [A-256](#)
 issue stop condition (TWISTOP), [A-259](#)
 lost arbitration (TWILOST), [A-261](#)
 master address length (TWIMLEN),
[A-258](#)
 master mode enable (TWIMEN), [A-258](#)
 master transfer direction (TWIMDIR),
[A-258](#)
 master transfer in progress
 (TWIMPORG), [A-261](#)
 not acknowledged (TWINAK), [A-256](#)
 repeat START (TWIRSTART), [A-259](#)
 serial clock override (TWISCLOVR),
[A-259](#)
 serial clock sense (TWISCLSEN), [A-262](#)
 serial data override (TWISDAOVR),
[A-259](#)
 serial data sense (TWISDASEN), [A-262](#)
 slave address length (TWISLEN), [A-255](#)
 slave enable (TWISEN), [A-255](#)
 slave transmit data valid (TWIDVAL),
[A-255](#)
 TWI controller registers
 clock divider (TWIDIV), [A-254](#)
 RXTWI16 (16-bit receive FIFO)
 register, [21-14](#)
 RXTWI8 (8-bit receive FIFO), [21-13](#)
 TWIMADDR (master mode address),
[A-260](#)

TWI controller registers (*continued*)
 TWIMCTL (master mode control),
[A-257](#)
 TWIMSTAT (master mode status),
[A-260](#)
 TWISADDR (slave mode address),
[A-256](#)
 TWISCTL (slave mode control), [A-255](#)
 TWISSTAT (slave mode status), [A-256](#)
 TXTWI16 (16-bit transmit FIFO),
[21-12](#)
 TXTWI8 (8-bit transmit FIFO), [21-12](#)
 two channel mode (S/PDIF), [13-12](#)
 TXFLSH (flush transmit buffer) bit, [15-35](#),
[A-236](#)
 TXS_A (data buffer channel B status) bit,
[A-158](#), [A-161](#), [A-166](#)
 TXSPI, TXSPIB (SPI transmit buffer)
 registers, [A-239](#)
 TXSPI (SPI transmit buffer) register, [2-10](#),
[15-20](#), [15-30](#)
 TXSPx (serial port transmit buffer)
 registers, [2-10](#)
 TXSR (SPI transmit shift) register, [15-8](#)
 TX_UACEN (DMA transmit buffer
 enable) bit, [20-5](#)

U

UART, [20-1](#)
 baud rate, [20-10](#), [20-11](#)
 baud rate examples, [20-5](#)
 block diagram, [20-7](#)
 chained DMA, [2-15](#), [20-14](#)
 core transfers, [20-12](#)
 data ready flag, [20-12](#)
 data word, [20-10](#)
 divisor, [20-4](#), [A-249](#)
 divisor reset, [20-5](#)
 DMA transfers, [20-13](#)
 sampling clock period, [20-12](#)

Index

- UART *(continued)*
- sampling point, 20-11
 - standard, 20-1
 - timers, 20-6
 - UART bits
 - 9-bit RX enable (RX9), A-250
 - 9-bit TX enable (TX9), A-250
 - address detect enable (UARTAEN), A-250
 - DMA TX/RX control, A-251
 - DMA TX/RX status, A-252
 - enable receive buffer full interrupt (UARTRBFIE), A-246
 - enable transmit buffer empty interrupt (UARTTBEIE), A-246
 - interrupt enable, A-247
 - pack data, 20-8
 - packing enable (PACK), A-250
 - pin status (UARTPSTx), A-251
 - program controlled interrupt bit (PCI), 20-22
 - synch data packing in RX (UARTPKSYN), A-250
 - THR register empty (UARTTHRE), A-245
 - UARTNOINT (pending interrupt), A-248
 - UARTSTAT (interrupt), A-248
 - UART registers
 - divisor latch register (UARTxDLL), 20-4, A-249
 - divisor latch (UARTxDLH), 20-4, A-249
 - interrupt enable register (UARTxIER), A-246
 - interrupt identification register (UARTxIIR), A-247
 - line control register (UARTxLCR), A-243
 - line status register (UARTxLSR), A-245
 - receive buffer register (UARTxRBR), 20-11
 - transmit holding (UARTxTHR), 20-10
 - transmit shift register (UART_TSР), 20-10
 - UARTxDLH (divisor latch register), 20-4, A-249
 - UARTxDLL (divisor latch register), 20-4, A-249
 - UARTxIER (interrupt enable register), A-246
 - UARTxIIR (interrupt identification register), A-247
 - UARTxLCR (line control register), A-243
 - UARTxLSR (line status register), A-245
 - UARTxRBR (receive buffer register), 20-11
 - UARTxTHR (transmit holding register), 20-10
 - UARTxTSР (transmit shift register), 20-10
- V
- VCO
- bypass clock, 22-7
 - clock, 22-5
 - examples, clock management, 22-15
 - output clock, 22-5
- W
- wait states, enabling (WS bit), A-22, A-49
- warnings and cautions
- DMA transfers, 2-28
 - I/O processor, 2-28
 - SPORTs, 10-42
- watchdog function, timer, 16-19
- watchdog timer
- clocking, 19-4

- clock pin (WDT_CLKIN), [19-4](#)
- pin descriptions, [19-3](#)
- register descriptions, [19-3](#)
- starting, [19-8](#)
- WDTH_CAP (width capture) mode, [16-2](#),
[16-12](#), [19-2](#)
- window processing, IIR, [6-61](#)
- word length, [10-10](#)
- word length (SLEN) bits, [10-29](#), [10-31](#)
- word packing enable (packing 16-bit to
32-bit words), [A-154](#), [A-159](#), [A-163](#)
- write-1-to-clear bit description (W1C), [A-4](#)
- write-only bit description (WO), [A-4](#)
- write-only-to-clear bit description (WOC),
[A-4](#)

Index