

ADSP-TS101 TigerSHARC® Processor Hardware Reference

Revision 1.1, May 2004

Part Number
82-001996-01

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



Copyright Information

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

Printed in the USA.

Disclaimer

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

Trademark and Service Mark Notice

The Analog Devices logo, EZ-ICE, SHARC, TigerSHARC, and the TigerSHARC logo are registered trademarks of Analog Devices, Inc.

VisualDSP++, Static Superscalar, and EZ-KIT Lite are trademarks of Analog Devices, Inc.

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

CONTENTS

PREFACE

Purpose of This Manual	xix
Intended Audience	xix
Manual Contents	xx
Additional Literature	xxii
What's New in This Manual	xxii
Technical or Customer Support	xxii
Processor Family	xxiii
Product Information	xxiii
DSP Product Information	xxiii
Product Related Documents	xxiv
Technical Publications Online or on the Web	xxiv
Printed Manuals	xxv
VisualDSP++ and Tools Manuals	xxv
Hardware Manuals	xxvi
Data Sheets	xxvi

Recommendations for Improving Our Documents	xxvi
Conventions	xxvii

INTRODUCTION

DSP Architecture	1-6
Compute Blocks	1-8
Arithmetic Logic Unit (ALU)	1-9
Multiply Accumulator (Multiplier)	1-11
Bit Wise Barrel Shifter (Shifter)	1-11
Integer Arithmetic Logic Unit (IALU)	1-12
Program Sequencer	1-13
Quad Instruction Execution	1-14
Relative Addresses for Relocation	1-15
Nested Call and Interrupt	1-15
Context Switching	1-15
Internal Memory and Other Internal Peripherals	1-16
Internal Buses	1-16
Internal Transfer	1-17
Data Accesses	1-17
Quad Data Access	1-17
Scalability and Multiprocessing	1-18
External Port	1-19
External Bus and Host Interface	1-19
External Memory	1-19
Multiprocessing	1-20

Host Interface	1-21
DMA Controller	1-22
Link Ports	1-23
Miscellaneous	1-23
Timers	1-23
Clock Domains	1-23
Booting	1-24
Emulation and Test Support	1-24
Programming Model	1-25

MEMORY AND REGISTER MAP

Memory Access Features	2-1
Host Address Space	2-2
External Memory Bank Space	2-4
Multiprocessor Space	2-5
Internal Address Space	2-6
Internal Memory Access	2-7
Broadcast Write	2-8
Merged Distribution	2-8
Broadcast Distribution	2-8
Data Alignment Buffer	2-9
Register Access Features	2-9
Register Space	2-9
Compute Block Register Files	2-10
Merged Access	2-11

Broadcast Transfer	2-11
Unmapped Compute Block Registers	2-12
Global Registers—XSTAT/YSTAT	
Compute Block Status Registers	2-13
ALU Registers	2-13
Multiplier Registers	2-13
Shifter Registers	2-13
Communications Logic Unit (CLU) Registers	2-13
IALU Registers	2-14
Register J31 – JSTAT	2-15
Register K31 – KSTAT	2-15
Sequencer Register Groups	2-15
Sequencer Control Register – SQCTL	2-16
Sequencer Control Register Set Bits – SQCTLST	2-16
Sequencer Control Register Clear Bits – SQCTLCL	2-16
Sequencer Status Register – SQSTAT	2-16
SFREG Register	2-16
ILAT Registers	2-19
IMASK Register	2-19
PMASK Register	2-19
Interrupt Vector Table Register Groups	2-20
Debug Register Groups	2-21
Performance Monitor Counter – PRFCNT	2-23
Performance Monitor Mask – PRFM	2-23
Watchpoint Control – WP0CTL, WP1CTL and WP2CTL	2-25

Watchpoint Status – WP0STAT, WP1STAT and WP2STAT	2-27
Cycle Counters – CCNT0 and CCNT1	2-29
Trace Buffer – TRCB0 to TRCB7 and TRCBPTR	2-29
Watchpoint Address Pointers – WP0L, WP1L, WP2L, WP0H, WP1H and WP2H	2-29
External Port Registers	2-30
Bus Control/Status (BIU) Register Group	2-30
SYSTAT/SYSTATCL Register	2-31
SYSCON Register (DMA 0x180480)	2-34
SDRCON (SDRAM Configuration) (DMA 0x180484)	2-36
BUSLK System Control	2-38
BMAX Current Value	2-39
BMAX Register	2-39
External Port Configuration and Status Registers	2-41
DMA Registers	2-41
External Port DMA Register	2-43
AutoDMA Registers	2-43
AutoDMA0 Register	2-44
AutoDMA1 Register	2-44
Link Port Transmit DMA Register	2-45
Link Port Receive DMA Register	2-45
DMA Control and Status Register	2-46
Link Registers	2-47
Link Port Control and Status Register	2-47
Link Port Receive and Transmit Buffers	2-48

CORE CONTROLS

Clock Inputs	3-1
Operation Modes	3-2
Emulation Mode	3-3
Supervisor Mode	3-4
User Mode	3-5
Low Power Mode	3-6
Entering Low Power Mode	3-6
Single Processor System	3-6
Multiprocessing Systems	3-7
Return to Normal Operation	3-8
Flag Pins	3-8
Timers	3-9
Timer Registers	3-9
Timer Operations	3-9
Timer 0 Output Pin	3-10

INTERRUPTS

Interrupt I/O Pins	4-2
Interrupt Vector Table	4-2
Interrupt Types	4-3
Level or Edge Interrupts	4-3
Interrupts Generated by On-Chip Modules	4-4
Timers	4-4

Link Interrupts	4-5
DMA Interrupts	4-5
Interrupt Pins (IRQ)	4-6
Vector Interrupt (VIRPT)	4-7
Bus Lock Interrupt	4-7
Hardware Error Operations	4-8
Software Exceptions	4-9
Emulation Debug	4-10
Other Interrupt Registers	4-11
ILAT Register	4-11
IMASK Register	4-12
PMASK Register	4-13
Interrupt Service	4-14
Interrupt Handling	4-19
Returning From Interrupt	4-22
Exceptions	4-23

CLUSTER BUS

External Bus Features	5-2
Bus Interface I/O Pins	5-3
Processor Microarchitecture	5-3
SYSCON Programming	5-11
Bus Width	5-14
Slow Device Protocol	5-14
Pipelined Protocol	5-15

Initial Value	5-15
TigerSHARC Pipelined Interface	5-15
Control Signals	5-16
Basic Transaction	5-16
Pipelining Transactions	5-18
Wait Cycles	5-23
Slow Device Protocol	5-27
EPROM Interface	5-31
Flyby Transactions	5-34
Multiprocessing	5-38
Bus Arbitration Protocol	5-41
Core Priority Access (CPA)	5-43
DMA Priority Access (DPA)	5-43
Bus Fairness — BMAX	5-47
Bus Lock	5-47
Host Interface	5-48
Backoff	5-50

SDRAM INTERFACE

SDRAM I/O Pins	6-7
SDRAM Physical Connection	6-8
Internal TigerSHARC processor Address and SDRAM Physical Connection	6-11
SDRAM Programming	6-19
SDRAM Control Register (SDRCON)	6-19

SDRAM Enable	6-22
Selecting the CAS Latency Value (CL)	6-22
Setting the SDRAM Buffering Option (Pipeline depth)	6-23
Selecting the SDRAM Page Size (Page Boundary)	6-25
Setting the Refresh Counter Value (Refresh Rate)	6-25
Selecting the Precharge to RAS Delay (tRP)	6-25
Selecting the RAS to Precharge Delay (tRAS)	6-26
Setting the SDRAM Power-Up Mode (Init Sequence)	6-27
SDRAM Interface Throughput	6-27
Multiprocessing Operation	6-28
Understanding DQM Operation	6-29
Powering Up After Reset	6-29
SDRAM Controller Commands	6-30
Mode Register Set (MRS) Command	6-30
Precharge (PRE) Command	6-31
Terminating Read/Write Cycles	6-32
Precharging	6-32
Bank Active (ACT) Command	6-33
Read Command	6-34
Bus Width = 64	6-36
Bus Width = 32	6-37
Write Command	6-38
Bus Width = 64	6-40
Bus Width = 32	6-41

Refresh (REF) Command	6-42
Self-Refresh (SREF) Command	6-42
Programming Example	6-43

DIRECT MEMORY ACCESS

DMA Controller Features	7-7
Cluster Bus Transfers	7-7
AutoDMA Transfers	7-9
Link Transfers	7-9
Two-Dimensional DMA	7-10
Chained DMA	7-11
DMA Architecture	7-11
DMAR I/O Pins	7-12
Terminology	7-13
Setting Up DMA Transfers	7-14
DMA Transfer Control Block Registers	7-15
DMA Channel Control	7-15
Transfer Control Block (TCB) Registers	7-15
DIx Register	7-16
DXx Register	7-17
DYx Register	7-17
DPx Register	7-18
DMA Control and Status Registers	7-23
DMA Status Register (DSTAT/DSTATC)	7-23
DMA Control Registers	7-26

DCNT Register	7-26
DCNTST Register	7-28
DCNTCL Register	7-28
DMA Control Register Restrictions	7-28
Operand Length Setup (Len)	7-28
Count (XCOUNT and YCOUNT)	7-28
Type Setup – Links Transmit (Channels 4 to 7)	7-29
Type Setup – Links Receive (Channels 8 to 11)	7-29
Type Setup – EP (Channels 0 to 3)	7-30
Type Setup – AutoDMA (Channels 12, 13)	7-30
DMA Request	7-30
Alignments	7-30
Address Range	7-31
DMA Controller Operations	7-32
Link Port DMA Control	7-32
External Port DMA Control	7-32
AutoDMA Register Control	7-33
DMA Transfers	7-33
Internal Memory Buses	7-33
DMA Channels	7-34
DMA Memory Accesses	7-34
DMA Channel Prioritization	7-36
Internal Memory Bus Priority	7-36
DMA Channel Priority	7-36

Rotating Priority	7-39
DMA Chaining	7-41
Enabling and Disabling Chaining	7-42
Interrupting Chaining	7-42
Transfer Control Blocks and Chain Loading	7-43
Setting Up and Starting the Chain	7-43
Chain Insertion	7-44
Two-Dimensional DMA	7-45
Two-Dimensional DMA Channel Organization	7-45
Two-Dimensional DMA Operation	7-46
DMA Interrupts	7-48
Starting and Stopping DMA Sequences	7-48
Starting a DMA Sequence	7-48
Ending a DMA Sequence	7-49
Suspending a DMA Sequence	7-49
Resuming a DMA Sequence	7-49
External Port DMA	7-50
Internal and External Address Generation	7-50
External Port DMA Transfer Types	7-51
External to Internal Memory	7-51
Internal to External Memory	7-54
External I/O Device to External Memory (Flyby)	7-55
External Memory to External I/O Device (Flyby)	7-57
DMA Semaphores	7-59

Handshake Mode	7-61
Flyby Mode	7-61
Link Ports DMA	7-63
Link Ports DMA Transfer Types	7-63
Link Port to Internal/External Memory	7-63
Internal/External Memory to Link	7-63
Receiving Link Port to Link Port	7-64
DMA Throughput	7-66
Internal Memory DMA	7-67
External Memory DMA	7-67
DMA Operation on Boot	7-68

LINK PORTS

Link Architecture	8-2
Link I/O Pins	8-2
Transmitting and Receiving Data	8-4
DMA	8-5
Interrupts	8-7
Reset and Boot	8-7
Link Port Communication Protocol	8-8
Transmission Delays	8-15
Error Detection Mechanisms	8-17
Transmitter Error Detection	8-18
Receiver Error Detection	8-18
Control Register (LCTLx)	8-19

Status Register (LSTATx)	8-23
--------------------------------	------

DEBUG FUNCTIONALITY

Operating Modes	9-2
Debug Resources	9-2
Special Instructions	9-2
Watchpoints	9-3
Programming – Control and Address Pointer Registers	9-4
Watchpoint Operation	9-7
Watchpoint Status (WPiSTAT)	9-8
Instruction Address Trace Buffer (TBUF)	9-9
Performance Monitors	9-10
Cycle Counter (CCNT1–0)	9-11
Performance Monitor Mask – PRFM	9-11
Performance Monitor Counter	9-12
JTAG Functionality	9-12
Pins	9-12
JTAG Instruction Register	9-14
Data Registers	9-14

SYSTEM DESIGN

Overview	10-1
TigerSHARC Processor Pins	10-3
Pin Definitions	10-4
Strap Pin Function Descriptions	10-7

Pin Usage	10-8
Pin States At Reset	10-10
Power, Reset, and Clock Input Considerations	10-11
Power Supply Sequencing	10-11
Reset	10-11
Timers	10-11
Timer Interrupt and FLAG I/O Examples	10-12
Clock Description and Jitter	10-15
Clock Distribution	10-15
General High Speed Clock Distribution Issues	10-16
Reset and Boot	10-17
Booting	10-18
Software Development Tools Support for Booting	10-18
Selecting the Booting Mode	10-18
Handling BMS	10-19
Master Mode Boot	10-19
Slave Boot Mode	10-19
General Boot Scenario	10-19
No Boot Mode	10-20
Booting a Single TigerSHARC Processor	10-20
EPROM/Flash Device Boot	10-20
Host Boot	10-22
Link Port Boot	10-24
Booting a Multiprocessor System	10-26

Multiprocessor EPROM Booting	10-26
All DSPs Boot in Turn From a Single EPROM	10-26
A Single TigerSHARC Processor Boots Other Processors	10-28
Multiprocessor Host Booting	10-30
Memory Initialization During Boot	10-31
Multiprocessor Link Port Booting	10-32
DMA Operation on Boot	10-32
Boot EPROM to Internal Memory	10-33
EPROM TCB	10-34
Internal Memory TCB	10-35
Boot Link to Internal Memory	10-36
Boot AutoDMA Register to Internal Memory	10-37
JTAG Issues	10-38
Resources and References	10-38
Decoupling Capacitors and Ground Plane Recommendations	10-38
Signal Integrity	10-39
IBIS Models	10-39
Output Pin Drive Strength Control	10-39
ADSP-TS101 Processor EZ-KIT Lite	10-40
Recommended Reading References	10-40

PREFACE

Thank you for purchasing and developing systems using Digital Signal Processors (DSPs) from Analog Devices.

Purpose of This Manual

The *ADSP-TS101 TigerSHARC Processor Hardware Reference* provides architectural information about the TigerSHARC® processors. The architectural descriptions cover functional blocks, buses, and ports, including all features and processes that they support. For programming information, see the *ADSP-TS101 TigerSHARC Processor Programming Reference*. For timing, electrical, and package specifications, see the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*.

Intended Audience

This manual is intended for TigerSHARC processor system designers and programmers who are familiar with digital signal processing (DSP) concepts. Users should have a working knowledge of microcomputer technology and DSP related mathematics.

Manual Contents

This manual provides detailed information about the TigerSHARC processor in the following chapters:

- “Introduction”

This chapter provides an architectural overview of the TigerSHARC processor.

- “Memory and Register Map”

This chapter defines the memory map of the TigerSHARC processor. The memory space defines the location of each element on the TigerSHARC processor.

- “Core Controls”

This chapter discusses clocking inputs, including the three different types of operating modes in which the TigerSHARC processor can operate and the boot modes from which the TigerSHARC processor initiates.

- “Interrupts”

This chapter discusses the various types of interrupts supported by the TigerSHARC processor. Some of the interrupts are generated internally or externally.

- “Cluster Bus”

This chapter focuses on the external bus interface of the TigerSHARC processor, which includes the bus arbitration logic and the external address, data and control buses.

- “SDRAM Interface”

This chapter describes TigerSHARC interface to SDRAM devices, including programming issues and the SDRAM controller.

- “Direct Memory Access”

This chapter describes how the TigerSHARC processor’s on-chip DMA controller acts as a machine for transferring data without core interruption.

- “Link Ports”

This chapter describes how link ports provide point-to-point communications between TigerSHARC processors in a system. The Link ports can also be used to interface with any other device that is designed to work in the same protocol.

- “Debug Functionality”

This chapter describes features of the TigerSHARC processor that are useful for performing software debugging and services usually found in Operating System (OS) kernels.

- “System Design”

This chapter describes system features of the TigerSHARC processor. These include Power, Reset, Clock, JTAG, and Booting, as well as pin descriptions and other system level information.

This hardware reference is a companion document to the *ADSP-TS101 TigerSHARC Processor Programming Reference*.

Additional Literature

The following publications that describe the TigerSHARC processor can be ordered from any Analog Devices sales office:

- *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*
- *ADSP-TS101 TigerSHARC Processor Hardware Reference*
- *ADSP-TS101 TigerSHARC Processor Programming Reference*

What's New in This Manual

This is the first revision of the *ADSP-TS101 TigerSHARC Processor Hardware Reference*. In future revisions, this section will document additions and corrections from previous revisions of the book.

Technical or Customer Support

You can reach our TigerSHARC processor Customer Support in the following ways:

- E-mail development tools questions to dsptools.support@analog.com
- E-mail processor questions to dsp.support@analog.com
- Phone questions to 1800-ANALOGD
- Visit our World Wide Web site at <http://www.analog.com/dsp>
- Telex questions to 924491, TWX: 710-394-6577
- Cable questions to ANALOG NORWOODMASS

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

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

Processor Family

The name *TigerSHARC* refers to the family of Analog Devices 32-bit, floating-point digital signal processors (DSP). This processor family currently includes the following processors:

- ADSP-TS101

Product Information

You can obtain product information from Analog Devices Web site, from the product CD-ROM, or from printed documents/manuals.

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

DSP Product Information

For information on digital signal processors, visit our Web site at <http://www.analog.com/dsp>. It provides access to technical information and documentation, product overviews, and product announcements.

Product Information

You may also obtain additional information about Analog Devices and its products by:

- FAXing questions or requests for information to **1-781-461-3010** (North America) or **089/76 903-557** (Europe Headquarters)
- Accessing the Digital Signal Processing Division FTP site:
ftp ftp.analog.com or **ftp 137.71.23.21** or
ftp://ftp.analog.com

Product Related Documents

For information on product related development software, see these publications:

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

Technical Publications Online or on the Web

You can access TigerSHARC processor documentation in these ways:

- **Online Access using VisualDSP++ Installation CD-ROM**

Your VisualDSP++™ software distribution CD-ROM includes all of the listed VisualDSP++ software tool publications.

After you install VisualDSP++ software on your PC, select the **Help Topics** command on the VisualDSP++ **Help** menu, click the **Reference** book icon, and select **Online Manuals**. From this **Help** topic, you can open any of the manuals, which are either in HTML format or in Adobe Acrobat PDF format.

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

- **Web Access**

Use the Analog Devices technical publications Web site http://www.analog.com/industry/dsp/tech_doc/gen_purpose.html to access DSP publications, including data sheets, hardware reference books, instruction set reference books, and VisualDSP++ software documentation. You can view, download, or print in PDF format. Some publications are also available in HTML format.

Printed Manuals

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

VisualDSP++ and Tools Manuals

The VisualDSP++ and Tools manuals may be purchased through Analog Devices North/Nashua Customer Service at 1-781-329-4700; ask for a Customer Service representative. The manuals can be purchased only as a kit. For additional information, call 1-603-883-2430.

If you do not have an account with Analog Devices, you are referred to Analog Devices distributors. To get information on our distributors, log on to http://www.analog.com/world/corp_fin/sales_directory/distrib.html.

Product Information

Hardware Manuals

Hardware reference books and instruction set reference books can be ordered through the Literature Center or downloaded from the Analog Devices Web site. The phone number is **1-800-ANALOGD (1-800-262-5643)**. The books can be ordered by a title or by product number located on the back cover of each book.

Data Sheets

All data sheets can be downloaded from the Analog Devices Web site. As a general rule, any data sheets with a letter suffix (L, M, N, S) can be obtained from the Literature Center at **1-800-ANALOGD (1-800-262-5643)** or downloaded from the Web site. Data sheets without the suffix can be downloaded from the Analog Devices Web site **ONLY**—no hard copies are available. You can ask for the data sheet by title of a part or by product number.

If you want to have a data sheet faxed, the FAX number for that service is **1-800-446-6212**. Follow the prompts—a list of the data sheet code numbers will be faxed to you upon an automated request. You should always call the Literature Center first to find out if requested data sheets are available.

Recommendations for Improving Our Documents

Please send us your comments and recommendations on how to improve our manuals. You can contact us at:

- Software/Development Tools manuals
dsptools.support@analog.com
- Data sheets, Hardware and Instruction Reference Set manuals
dsp.support@analog.com

Conventions

The following table identifies and describes text conventions used in this manual.




 Note that additional conventions, which apply only to specific chapters, may appear throughout this document.

Table P-1. Notation Conventions

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 are delimited with a vertical bar; read the example as <i>this</i> or <i>that</i> . One or the other is required.
{this that}	Optional items in syntax descriptions appear within curly braces; read the example as an optional <i>this</i> or <i>that</i> .
[{{(S SU)}}]	Optional items for some lists may appear within parenthesis. If an option is chosen, the parenthesis must be used (for example, (S)). If no option is chosen, omit the parenthesis.
.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.
0xFBCD CBA9	Hexadecimal numbers use the 0x prefix and are typically shown with a space between the upper four and lower four digits.
b#1010 0101	Binary numbers use the b# prefix and are typically shown with a space between each four digit group.
	This symbol indicates a note that provides supplementary information on a related topic. In the online version of this book, the word Note appears instead of this symbol.

Conventions

Table P-1. Notation Conventions (Cont'd)

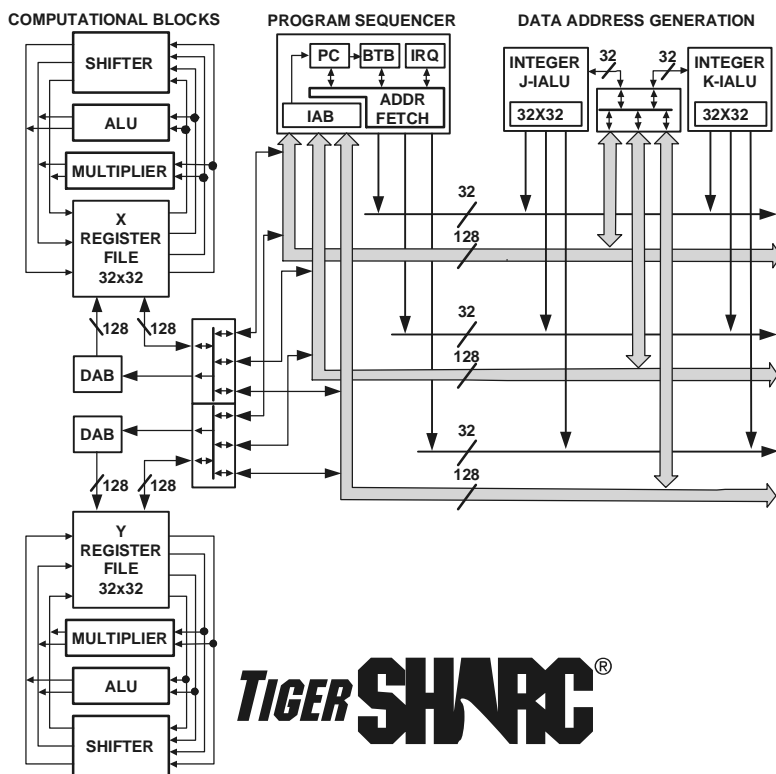
Example	Description
	This symbol indicates a warning that advises on an inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word Warning appears instead of this symbol.
LSB, MSB LSW, MSW	Abbreviations for Least Significant Bit and for Most Significant Bit. Abbreviations for Least Significant Word and for Most Significant Word.

1 INTRODUCTION

The *ADSP-TS101 TigerSHARC Processor Hardware Reference* contains architectural information required for designing TigerSHARC based systems. A separate document, the *ADSP-TS101 TigerSHARC Processor Programming Reference*, contains the instruction set description required for programming the TigerSHARC processor. In addition to this manual, designers should refer to the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet* for pin descriptions, and timing, electrical, and package specifications.

The TigerSHARC processor is a 32-bit, high performance DSP. The TigerSHARC processor sets a new standard of performance for digital signal processors, combining multiple computation units for floating-point and fixed-point processing as well as very wide word widths. The TigerSHARC processor maintains a “system-on-a-chip” scalable computing design philosophy, including 6M bits of on-chip SRAM, integrated I/O peripherals, on-chip SDRAM controller, a host processor interface, DMA controller, link ports, and shared bus connectivity for glueless multiprocessing.

In addition to providing unprecedented performance in DSP applications in raw MFLOPS and MIPS, the TigerSHARC processor boosts performance measures such as MFLOPS/Watt and MFLOPS/square inch in multiprocessing applications.



TIGER SHARC®

Figure 1-1. TigerSHARC Core Diagram

As shown in Figure 1-1 and Figure 1-2, the processor has the following architectural features:

- Dual computation blocks—X and Y—each consisting of a multiplier, ALU, shifter, and a 32-word register file
- Dual integer ALUs—J and K—each containing a 32-bit IALU and 32-word register file

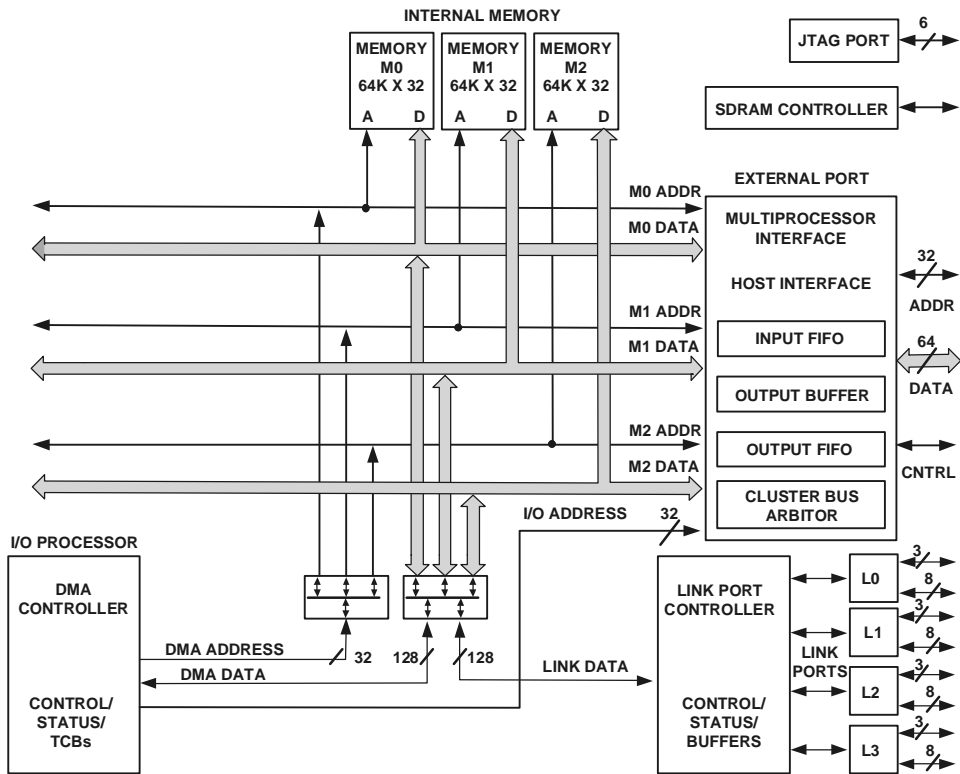


Figure 1-2. TigerSHARC I/O Peripherals Diagram

- Program sequencer—Controls the program flow and contains an instruction alignment buffer (IAB) and a branch target buffer (BTB)
- Three 128-bit buses providing high bandwidth connectivity between all blocks
- External port interface including the host interface, SDRAM controller, static pipelined interface, four DMA channels, four link ports (each with two DMA channels), and multiprocessing support

- 6M bits of internal memory organized as three blocks—M0, M1 and M2—each containing 16K rows and 128 bits wide (a total of 2M bit)
- Debug features
- JTAG Test Access Port

The TigerSHARC processor external port provides an interface to external memory, to memory-mapped I/O, to host processor, and to additional TigerSHARC processors. The external port performs external bus arbitration and supplies control signals to shared, global memory and I/O devices.

Figure 1-3 illustrates a typical single-processor system. A multiprocessor system is illustrated in Figure 1-4 on page 1-6 and is discussed later in “Scalability and Multiprocessing” on page 1-18.

The TigerSHARC processor includes several features that simplify system development. The features lie in three key areas:

- Support of IEEE floating-point formats
- IEEE 1149.1 JTAG serial scan path and on-chip emulation features
- Architectural features supporting high-level languages and operating systems

The features of the TigerSHARC architecture that directly support high-level language compilers and operating systems include:

- Simple, orthogonal instruction allowing the compiler to efficiently use the multi-instruction slots
- General-purpose data and IALU register files
- 32- and 40-bit floating-point and 8-, 16-, 32-, and 64-bit fixed-point native data types

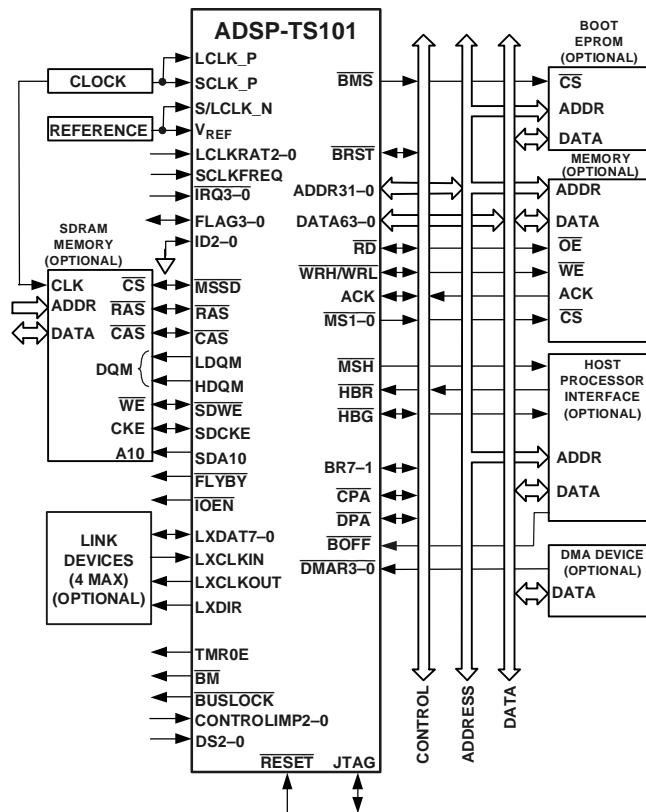


Figure 1-3. Single Processor Configuration

- Large address space
- Immediate address modify fields
- Easily supported relocatable code and data
- Fast save and restore of processor registers onto internal memory stacks

DSP Architecture

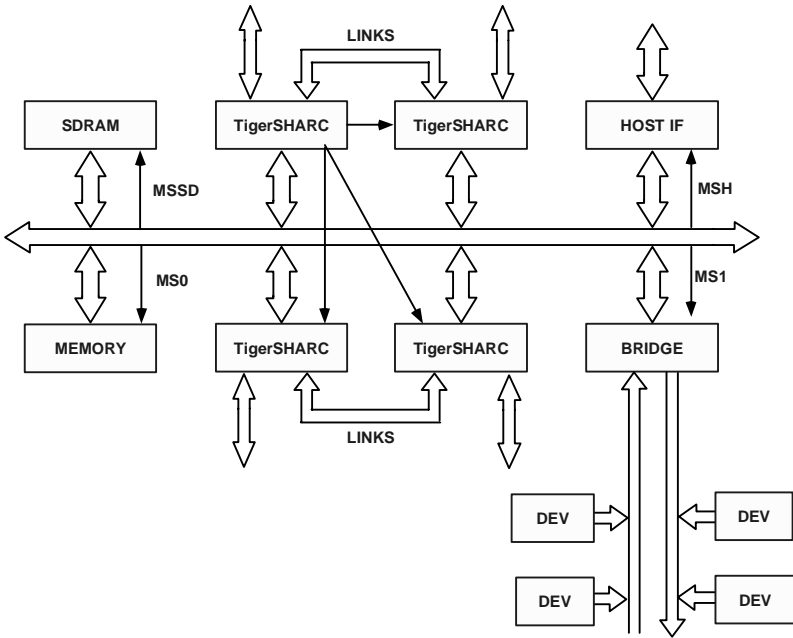


Figure 1-4. Multiprocessing Cluster Configuration

DSP Architecture

As shown in Figure 1-1 on page 1-2 and Figure 1-2 on page 1-3, the DSP architecture consists of two divisions: the DSP core (where instructions execute) and the I/O peripherals (where data is stored and off-chip I/O is processed). The following discussion provides a high-level description of the DSP core and peripherals architecture. More detail on the core appears in other sections of this reference.

High performance is facilitated by the ability to execute up to four 32-bit wide instructions per cycle. The TigerSHARC processor uses a variation of a *Static Superscalar*TM architecture to allow the programmer to specify

which instructions are executed in parallel in each cycle. The instructions do not have to be aligned in memory so that program memory is not wasted.

The 6M bit internal memory is divided into three 128-bit wide memory blocks. Each of the three internal address/data bus pairs connect to one of the three memory blocks. The three memory blocks can be used for triple accesses every cycle where each memory block can access up to four, 32-bit words in a cycle.

The external port cluster bus can be configured to be either 32 or 64 bits wide. The high I/O bandwidth complements the high processing speeds of the core. To facilitate the high clock rate, the TigerSHARC processor uses a pipelined external bus with programmable pipeline depth for interprocessor communications and for Synchronous DRAM (SDRAM).

The four link ports support point-to-point high bandwidth data transfers. Link ports have hardware-supported two-way communication.

The processor operates with a two cycle arithmetic pipeline. The branch pipeline is two to six cycles. A branch target buffer (BTB) is implemented to reduce branch delay. The two identical computation units support floating-point as well as fixed-point arithmetic.

During compute intensive operations, one or both integer ALUs compute or generate addresses for fetching up to two quad operands from two memory blocks, while the program sequencer simultaneously fetches the next quad instruction from the third memory block. In parallel, the computation units can operate on previously fetched operands while the sequencer prepares for a branch.

While the core processor is doing the above, the DMA channels can be replenishing the internal memories in the background with quad data from either the external port or the link ports.

The processing core of the TigerSHARC reaches exceptionally high performance through using these features:

- Computation pipeline
- Dual computation units
- Execution of up to four instructions per cycle
- Access of up to eight words per cycle from memory

The two computation units (compute blocks) perform up to 6 floating-point or 24 fixed-point operations per cycle.

Each multiplier and ALU unit can execute four 16-bit fixed-point operations per cycle, using Single-Instruction, Multiple-Data (SIMD) operation. This operation boosts performance of critical imaging and signal processing applications that use fixed-point data.

Compute Blocks

The TigerSHARC processor core contains two computation units called *compute blocks*. Each compute block contains a register file and three independent computation units—an ALU, a multiplier, and a shifter. For meeting a wide variety of processing needs, the computation units process data in several fixed- and floating-point formats listed here and shown in Figure 1-5:

- **Fixed-point format**
These include 64-bit long-word, 32-bit normal word, 16-bit short word, and 8-bit byte word. For short word fixed-point arithmetic,

quad parallel operations on quad-aligned data allow fast processing of array data. Byte operations are also supported for octal-aligned data.

- **Floating-point format**

These include 32-bit normal word and 40-bit extended word. Floating-point operations are single or extended precision. The normal word floating-point format is the standard IEEE format, and the 40-bit extended-precision format occupies a double word (64 bits) with eight additional LSBs of mantissa for greater accuracy.

Each compute block has a general-purpose, multi-port, 32-word data register file for transferring data between the computation units and the data buses and storing intermediate results. All of these registers can be accessed as single-, dual-, or quad-aligned registers. For more information on the register file, see “Compute Block Register Files” on page 2-10.

Arithmetic Logic Unit (ALU)

The ALU performs arithmetic operations on fixed-point and floating-point data and logical operations on fixed-point data. The source and destination of most ALU operations is the compute block register file.

On the TigerSHARC processor, the ALU includes a special sub-block, which is referred to as the *communications logic unit (CLU)*. The CLU instructions are designed to support different algorithms used for communications applications. The algorithms that are supported by the CLU instructions are:

- Viterbi Decoding
- Turbo-code Decoding
- De-spreading for code-division-multiple-access (CDMA) systems

DSP Architecture

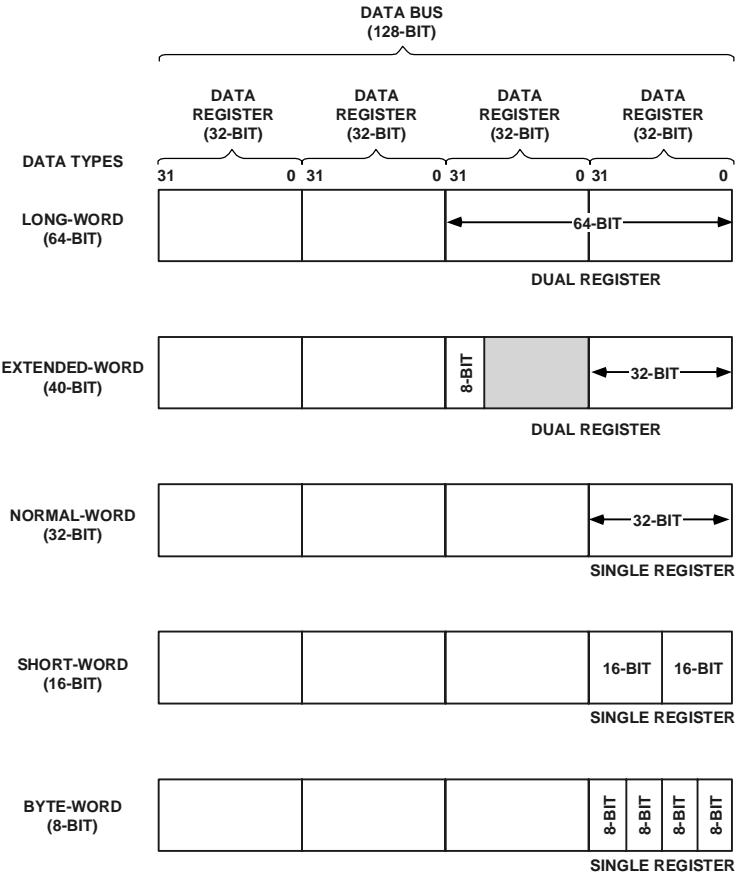


Figure 1-5. Word Format Definitions¹

¹ The TigerSHARC processor internal data buses are 128 bits (one quad-word) wide. In a quad-word, the DSP can move 16 byte words, 8 short words, 4 normal words, or 2 long-words over the bus at the same time.

Multiply Accumulator (Multiplier)

The multiplier performs fixed-point or floating-point multiplication and fixed-point multiply/accumulate operations. The multiplier supports several data types in fixed- and floating-point. The floating-point formats are float and float-extended, as in the ALU. The source and destination of most operations is the compute block register file.

The TigerSHARC processor multiplier supports complex multiply-accumulate operations. Complex numbers are represented by a pair of 16-bit short words within a 32-bit word. The least significant bits (LSBs) of the input operand represent the real part, and the most significant bits (MSBs) of the input operand represent the imaginary part.

For more information on the multiplier, see “Multiplier Registers” on page 2-13.

Bit Wise Barrel Shifter (Shifter)

The shifter performs logical and arithmetic shifts, bit manipulation, field deposit, and field extraction. The shifter operates on one 64-bit, one or two 32-bit, two or four 16-bit, and four or eight 8-bit fixed-point operands. Shifter operations include:

- Shifts and rotates from off-scale left to off-scale right
- Bit manipulation operations, including bit set, clear, toggle and test
- Bit field manipulation operations, including field extract and deposit, using register `BFOTMP` (which is internal to the shifter)
- Bit FIFO operations to support bit streams with fields of varying length

For more information on the shifter, see “Shifter Registers” on page 2-13.

Integer Arithmetic Logic Unit (IALU)

The IALUs can execute standard standalone ALU operations on IALU register files. The IALUs also provide memory addresses when data is transferred between memory and registers. The DSP has dual IALUs (the J-IALU and the K-IALU) that enable simultaneous addresses for multiple operand reads or writes. The IALUs allow computational operations to execute with maximum efficiency because the computation units can be devoted exclusively to processing data.

Each IALU has a multiport, 32-word register file. Operations in the IALU are not pipelined. The IALUs support pre-modify with no update and post-modify with update address generation. Circular data buffers are implemented in hardware. The IALUs support the following types of instructions:

- Regular IALU instructions
- Move Data instructions
- Load Data instructions
- Load/Store instructions with register update
- Load/Store instructions with immediate update

For indirect addressing (instructions with update), one of the registers in the register file can be modified by another register in the file or by an immediate 8- or 32-bit value, either before (pre-modify) or after (post-modify) the access. For circular buffer addressing, a length value can be associated with the first four registers to perform automatic modulo addressing for circular data buffers; the circular buffers can be located at arbitrary boundaries in memory. Circular buffers allow efficient implementation of delay lines and other data structures, which are commonly used in digital filters and Fourier transformations. The TigerSHARC processor circular buffers automatically handle address pointer wraparounds, reducing overhead and simplifying implementation.

The IALUs also support bit-reverse addressing, which is useful for the Fast Fourier Transform FFT algorithm. Bit-reverse addressing is implemented using a reverse carry addition that is similar to regular additions, but the carry is taken from the upper bits and is driven into lower bits.

The IALU provides flexibility in moving data as single, dual, or quad-words. Every instruction can execute with a throughput of one per cycle. IALU instructions execute with a single cycle of latency while computation units have two cycles of latency. Normally, there are no dependency delays between IALU instructions, but if there are, three or four cycles of latency can occur.

For more information on the IALUs, see “IALU Registers” on page 2-14.

Program Sequencer

The program sequencer supplies instruction addresses to memory and, together with the IALUs, allows computational operations to execute with maximum efficiency. The sequencer supports efficient branching using the branch target buffer (BTB), which reduces branch delays for conditional and unconditional instructions.

The TigerSHARC processor achieves its fast execution rate by means of an eight-cycle pipeline.

Two stages of the sequencer’s pipeline actually execute in the computation units. The computation units perform single-cycle operations with a two-cycle computation pipeline, meaning that results are available for use two cycles after the operation is begun. Hardware causes a stall if a result is not available in a given cycle (register dependency check). Up to two computation instructions per compute block can be issued in each cycle, instructing the ALU, multiplier, or shifter to perform independent, simultaneous operations.

DSP Architecture

The TigerSHARC processor has four general-purpose external interrupts, $\overline{IRQ3-0}$. The processor also has internally generated interrupts for the two timers, DMA channels, link ports, arithmetic exceptions, multiprocessor vector interrupts, and user-defined software interrupts. Interrupts can be nested through instruction commands. Interrupts have a short latency and do not abort currently executing instructions. Interrupts vector directly to a user-supplied address in the interrupt table register file, removing the overhead of a second branch.

The branch penalty in a deeply pipelined processor such as the TigerSHARC processor can be compensated for by the use of a branch target buffer (BTB) and branch prediction. The branch target address is stored in the BTB. When the address of a jump instruction, which is predicted by the user to be taken in most cases, is recognized (the tag address), the corresponding jump address is read from the BTB and is used as the jump address on the next cycle. Thus the latency of a jump is reduced from three to six wasted cycles to zero wasted cycles. If this address is not stored in the BTB, the instruction must be fetched from memory.

Other instructions also use the BTB to speed up these types of branches. These instructions are interrupt return, call return, and computed jump instructions.

Immediate extensions are associated with IALU or sequencer (control flow) instructions. These instructions are not specified by the programmer, but are implied by the size of the immediate data used in the instructions.

For more information on the sequencer, BTB, and immediate extensions, see “Sequencer Register Groups” on page 2-15.

Quad Instruction Execution

The TigerSHARC processor can execute up to four instructions per cycle from a single memory block, due to the 128-bit wide access per cycle. The ability to execute several instructions in a single cycle derives from a *Static*

Superscalar architectural concept. This is not strictly a superscalar architecture because the instructions executed in each cycle are specified in the instruction by the programmer or by the compiler, and not by the chip hardware. There is also no instruction reordering. Register dependencies are, however, examined by the hardware and stalls are generated where appropriate. Code is fully compacted in memory and there are no alignment restrictions for instruction lines.

Relative Addresses for Relocation

Most instructions in the TigerSHARC processor support PC relative branches to allow code to be relocated easily. Also, most data references are *register relative*, which means they allow programs to access data blocks relative to a base register.

Nested Call and Interrupt

Nested call and interrupt return addresses (along with other registers as needed) are saved by specific instructions onto the on-chip memory stack, allowing more generality when used with high-level languages. Non-nested calls and interrupts do not need to save the return address in internal memory, making these more efficient for short, non-nested routines.

Context Switching

The TigerSHARC processor provides the ability to save and restore up to eight registers per cycle onto a stack in two internal memory blocks when using load/store instructions. This fast save/restore capability permits efficient interrupts and fast context switching. It also allows the TigerSHARC processor to dispense with on-chip PC stack or alternate registers for register files or status registers.

Internal Memory and Other Internal Peripherals

The on-chip memory consists of three blocks of 2M bits each. Each block is 128 bits (four words) wide, thus providing high bandwidth sufficient to support both computation units, the instruction stream and external I/O, even in very intensive operations. The TigerSHARC processor provides access to program and two data operands without memory or bus constraints. The memory blocks can store instructions and data interchangeably.

Each memory block is organized as 64K words of 32 bits each. The accesses are pipelined to meet one clock cycle access time needed by the core, DMA, or by the external bus. Each access can be up to four words. Memories (and their associated buses) are a resource that must be shared between the compute blocks, the IALUs, the sequencer, the external port, and the link ports. In general, if during a particular cycle more than one unit in the processor attempts to access the same memory, one of the competing units is granted access, while the other is held off for further arbitration until the following cycle—For more information, see “Bus Arbitration Protocol” on page 5-41. This type of conflict only has a small impact on performance due to the very high bandwidth afforded by the internal buses.

An important benefit of large on-chip memory is that by managing the movement of data on and off chip with DMA, a system designer can realize high levels of determinism in execution time. Predictable and deterministic execution time is a central requirement in DSP and real-time systems.

Internal Buses

The processor core has three buses, each one connected to one of the internal memories. These buses are 128 bits wide to allow up to four instructions, or four aligned data words, to be transferred in each cycle on each bus. On-chip system elements also use these buses to access memory.

Only one access to each memory block is allowed in each cycle, so DMA or external port transfers must compete with core accesses on the same block. Because of the large bandwidth available from each memory block, not all the memory bandwidth can be used by the core units, which leaves some memory bandwidth available for use by the processor's DMA processes or by the bus interface to serve other DSPs bus master transfers to the TigerSHARC processor's memory.

Internal Transfer

Most registers of the TigerSHARC processor are classified as universal registers (Uregs). Instructions are provided for transferring data between any two Uregs, between a Ureg and memory, or for the immediate load of a Ureg. This includes control registers and status registers, as well as the data registers in the register files. These transfers occur with the same timing as internal memory load/store.

Data Accesses

Each move instruction specifies the number of words accessed from each memory block. Two memory blocks can be accessed on each cycle because of the two IALUs.

Quad Data Access

Instructions specify whether one, two, or four words are to be loaded or stored. Quad-words¹ can be aligned on a quad-word boundary and long-words² aligned on a long-word boundary. This, however, is not necessary when loading data to computation units because a data alignment buffer (DAB) automatically aligns quad-words that are not aligned in memory.

¹ A memory quad-word is comprised of four 32-bit words or 128 bits of data.

² A memory long-word is comprised of two 32-bit words or 64 bits of data.

Up to four data words from each memory block can be supplied to each computation unit, meaning that new data is not required on every cycle thus leaving alternate cycles for I/O to the memories. This is beneficial in applications with high I/O requirements since it allows the I/O to occur without degrading core processor performance.

Scalability and Multiprocessing

The TigerSHARC processor, like the related Analog Devices product the SHARC processor, is designed for multiprocessing applications. The primary multiprocessing architecture supported is a cluster of up to eight TigerSHARC processors that share a common bus, a global memory, and an interface to either a host processor or to other clusters. In large multiprocessing systems, this cluster can be considered an element and connected in configurations such as torroid, mesh, tree, crossbar, or others. The user can provide a personal interconnect method or use the on-chip communication ports.

The TigerSHARC processor improves on most of the multiprocessing capabilities of the SHARC processor and enhances the data transfer bandwidth. These capabilities include:

- On-chip bus arbitration for glueless multiprocessing
- Globally accessible internal memory and registers
- Semaphore support
- Powerful, in-circuit multiprocessing emulation

External Port

External Bus and Host Interface

The TigerSHARC processor external port (EP) provides an interface between the core processor and the 32/64-bit parallel external bus. The external port contains FIFOs that maintain the throughput of an external bus that is shared with multiple processors and peripherals—each of which may operate at speeds other than that of the core.

The most effective way to access external data in the TigerSHARC processor is through the DMA. This runs in the background, allowing the core to continue processing while new data is read in or processed data is written out. Multiple DMA data streams can occur simultaneously, and the use of FIFOs helps to maintain throughput in the system.

Burst accesses are provided through the $\overline{\text{BRST}}$ pin, which allows a slave device on the bus to accept the first address and then automatically increment that address as successive data words arrive.

External Memory

The TigerSHARC processor external port provides the processor interface to off-chip memory and peripherals. The off-chip memory and peripherals are included in the TigerSHARC processor unified address space. The separate on-chip buses are multiplexed at the external port to create an external system bus with a single 32-bit address bus and a single 64-bit data bus. External memory and devices can be either 32 or 64 bits wide. The TigerSHARC processor automatically packs external data into either 32-, 64-, or 128-bit word widths, the latter being more efficient for reducing memory access conflicts.

On-chip decoding of high order address lines (to generate memory block select signals) facilitates addressing of external memory devices. Separate control lines are also generated for simplified addressing of page mode DRAM.

DSP Architecture

The TigerSHARC processor uses the address on the external port bus to pipeline the data. This allows interfacing to synchronous DRAM and speeds up interprocessor accesses. An option allows asynchronous operation for slower devices.

External data can be accessed by DMA channels or by the core. For core accesses, the read latency can be significant—eight or more cycles. The core provides I/O buffering by stalling if the data is accessed before the data is loaded in a universal register (Ureg).

Programmable memory wait states permit peripherals with different pipeline delay cycle, access, hold, and disable time requirements.

External shared memory resources are assigned between processors by using semaphore operations.

Multiprocessing

The TigerSHARC processor offers features tailored to multiprocessing systems:

- The unified address space allows direct interprocessor accesses of each TigerSHARC processor internal memory and resources.
- Distributed bus arbitration logic is included on chip for glueless connection of systems containing up to eight TigerSHARC processors and a host processor.
- Bus arbitration rotates, except for host requests that always hold the highest priority.
- Processor bus lock allows indivisible read-modify-write sequences for semaphores.

- A vector interrupt capability is provided for interprocessor commands.
- Broadcast writes allow simultaneous transmissions of data to all TigerSHARC processors.

Host Interface

Connecting a host processor to a cluster of TigerSHARC processors is simplified by the memory-mapped nature of the interface bus and the availability of special host bus request signals.

A host that is able to access a pipelined memory interface can be easily connected to the parallel TigerSHARC processor bus. All the internal memory, Uregs, and resources within the TigerSHARC processor, such as the DMA control registers and the internal memory, are accessible to the host.

The host interface is through the TigerSHARC processor external address and data bus, with additional lines being provided for host control. The protocol is similar to the standard TigerSHARC processor pipelined bus protocol.

The host becomes bus master of the cluster by asserting the Host Bus Request ($\overline{\text{HBR}}$) signal. Host Bus Grant ($\overline{\text{HBG}}$) is returned by the TigerSHARC processors when the current master grants bus by asserting $\overline{\text{HBR}}$. The host interface is synchronous, and can be delayed a number of cycles to allow slow host access. The host can also access external memory directly.

All DMA channels are accessible to the host interface, allowing code and data transfers to be accomplished with low software overhead. The host can directly read and write the internal memory of the TigerSHARC processor and can access the DMA channel setup. Vector interrupt support is provided for efficient execution of host commands and burst-mode transfers.

DMA Controller

The TigerSHARC processor on-chip DMA controller allows zero-overhead data transfers without processor intervention. The TigerSHARC processor can simultaneously fetch instructions and access two memories for data without relying on data or instruction caches. The DMA controller operates independently of the processor core, supplying addresses for internal and external memory access. The DMA channels, therefore, are not part of the core processor from a programming point of view.

Both code and data can be downloaded to the TigerSHARC processor using DMA transfers, which can occur between the following.

- TigerSHARC processor internal memory and external memory, external peripherals or a host processor
- External memory and external peripheral devices
- External memory and link ports or between two link ports

Six DMA channels are available on the TigerSHARC processor for data transfers through the external port. Eight DMA channels are available for link data transfers (two per link).

Asynchronous off-chip peripherals can control any one of four DMA channels using DMA request lines (DMAR3-0). Other DMA features include flyby (for channel 0 only), interrupt generation upon completion of DMA transfers, and DMA chaining for automatically linked DMA transfers.

Link Ports

The TigerSHARC processor has four 8-bit link ports that provide additional I/O capabilities in multiprocessing systems. The link ports have the following characteristics.

- Link clock speed is selectable as either 1/8, 1/4, 1/3, or 1/2 of internal clock frequency.
- Link port data is packed into 128-bit words for DMA transfer to on- or off-chip memory.
- Each link port has its own buffer registers.
- Link port transfers are controlled by clock/acknowledge handshaking.
- Link ports support bidirectional transfer and flow through and transfers to/from the external port or other links.

Miscellaneous

Timers

The TigerSHARC processor has two programmable interval timers that provide periodic interrupt generation. When enabled, the timers decrement a 64-bit count register every cycle. When this count register reaches zero, the TigerSHARC processor generates an interrupt and asserts `TMROE` output (for timer zero only). The count register is automatically reloaded from a 64-bit period register and the count resumes immediately.

Clock Domains

There are two major clock domains in the TigerSHARC processor, driven by two input clocks—the local clock (`LCLK`) and the system clock (`SCLK`).

Booting

The AC specification and bus interface are defined in reference to the SCLK. (See the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet* for the full AC specification.) The internal SCLK is phase locked to the SCLK input by a Phase Locked Loop (PLL).

The LCLK is an input to the internal clock driver—CCLK. The CCLK is the internal clock of the core, internal buses, memory, links, and most of the chip's internal parts. The CCLK is generated by a PLL from LCLK and is phase locked. The LCLKRAT pins define the clock multiplication of LCLK to CCLK. The clock multiplication can be 2, 2.5, 3, 3.5, 4, 5, and 6.

Systems must connect both LCLK and SCLK to the same clock source. Using an integer LCLKRAT (2, 3, 4, 5, or 6) guarantees predictable cycle-by-cycle operation (important for SIMD operation and for fault tolerant systems).

Booting

The internal memory of the TigerSHARC processor can be loaded from an 8-bit EPROM using a boot mechanism at system power up. The TigerSHARC processor can also be booted using another master or through one of the link ports. Selection of the boot source is controlled by external pins.

Emulation and Test Support

The TigerSHARC processor supports the IEEE standard 1149.1 Joint Test Action Group (JTAG) standard for system test. This standard defines a method for serially scanning the I/O status of each component in a system. The JTAG serial port is also used by the TigerSHARC processor EZ-ICE® to gain access to the processor's on-chip emulation features.

Programming Model

Refer to the *ADSP-TS101 TigerSHARC Processor Programming Reference* for more detailed information.

2 MEMORY AND REGISTER MAP

This chapter describes the TigerSHARC processor memory and register map. For information on using registers for computations and memory for register loads and stores, see the *ADSP-TS101 TigerSHARC Processor Programming Reference*. For information on using registers for configuring the TigerSHARC processor's peripherals use the applicable chapters in this book.

Memory Access Features

The TigerSHARC processor has three internal memory blocks M0, M1, and M2 as shown in the system block diagram on Figure 1-2 on page 1-3. Each memory block consists of 2M bits of memory space, and is configured as 64k words each 32-bits in width. There are three separate internal 128-bit data buses, each connected to one of the memory blocks. Memory blocks can store instructions and data interchangeably, with one access per memory block per cycle. If the programmer ensures that program and data are in different memory blocks, then data access can occur at the same time as program fetch. Thus in one cycle, up to three 128-bit transfers can occur within the core (two data transfers, and one program instruction transfer).

The I/O Processor can use only one internal bus at a time, and the I/O Processor competes with the core for use of the internal bus. Therefore in one cycle, the processor can fetch four 32-bit instructions, and load or store 256 bits of data (four 64-bit words or eight 32-bit words or sixteen 16-bit words or thirty-two 8-bit words).

Memory Access Features

The TigerSHARC processor 32-bit address bus provides an address space of four gigawords. This address space is common to a cluster of TigerSHARC processors that share the same cluster bus. This chapter defines the memory map of each TigerSHARC processor in the system and indicates where the memory space defines the location of each element. The zones in the memory space are made up of the following regions.

- External memory bank space—the region for standard addressing of off-chip memory (including SDRAM, MB0, MB1, and Host)
- External multiprocessor space—the on-chip memory of all other TigerSHARC processors connected in a multiprocessor system
- Internal address space—the region for standard internal addressing

The global memory map is shown in Figure 2-1.

Host Address Space

The host address space is the space defined for the host when it is accessed as a slave. When referring to this space, the pipelined or asynchronous protocol is used according to the host bits in the SYSTAT register—for additional information on the SYSTAT/SYSTATCL register see Figure 2-9 on

Memory and Register Map

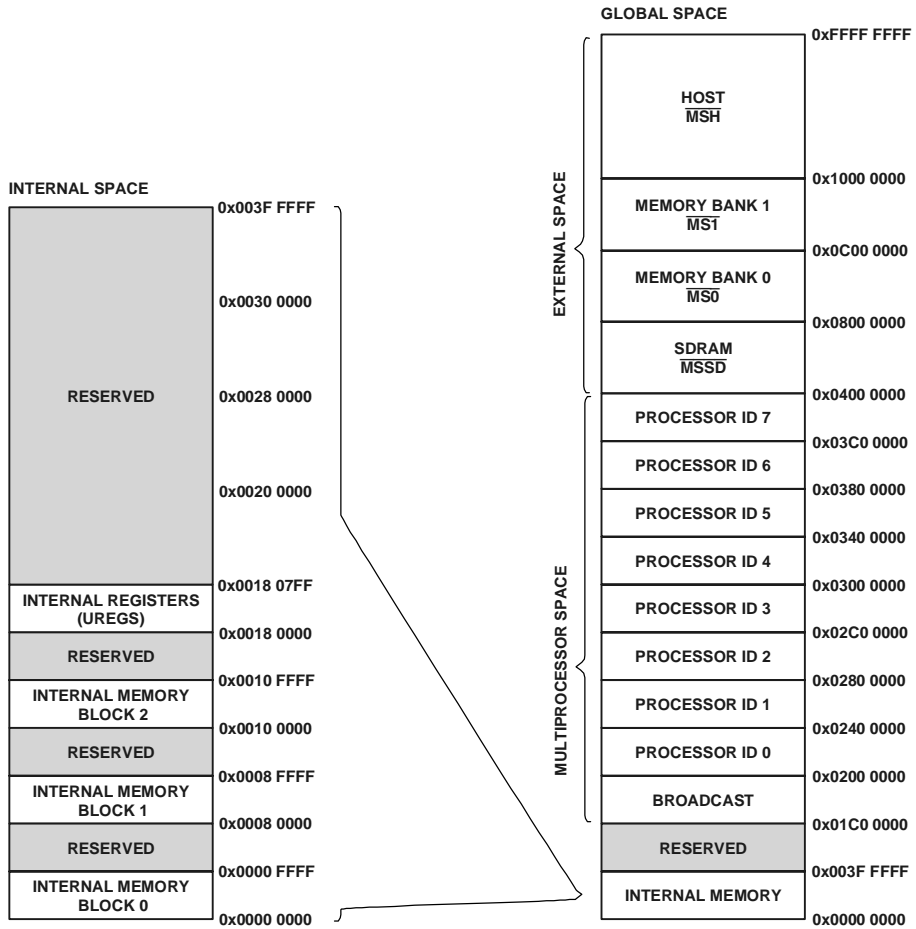


Figure 2-1. Global Memory Map

Memory Access Features

page 2-33. The backoff is also effective on this zone—see “Backoff” on page 5-50. The host address is 3.75 gigawords and is divided into two fields.

Table 2-1. Host Address

Bits	Name	Definition
ADDR27–0	Address	Address in host range
ADDR31–28	Host Select	Determines the type of address. As long as the value is not 0000, the address is directed to the host address space 0b0001 – 0b1111 (Host space)

External Memory Bank Space

Corresponds to off-chip memory and memory mapped I/O devices (SDRAM, I/O peripherals, and other standard memory devices).

Normal external accesses are split into three zones. One zone is for SDRAM; the access is executed in SDRAM protocol. The other two zones are identified by external memory select pins $\overline{MS0}$ and $\overline{MS1}$, where the access protocol is user configurable as pipelined or slow device protocol, and the parameters are defined by the `SYSCON` register value—see section “Bus Control/Status (BIU) Register Group” on page 2-30.

The external memory address is divided into three fields, as illustrated below.

Table 2-2. External Memory Bank Space

Bits	Name	Definition
ADDR25–0	Address	Address
ADDR27–26	MS	Memory Select—select the memory bank 00 – Internal or multiprocessor space 01 – \overline{MSSD} SDRAM 10 – $\overline{MS0}$ 11 – $\overline{MS1}$
ADDR31–28	Host Select	Determine the type of address. As long as the value is not 0000, the address is directed to the host address space 0b0001 – 0b1111(Host space)

Multiprocessor Space


The multiprocessing space maps the internal memory space of each TigerSHARC processor in the cluster into any other TigerSHARC processor. This allows one processor to easily write to and read from other processors in a multiprocessor system. Broadcast space allows write access to all TigerSHARC processors in the cluster. Each TigerSHARC processor in the cluster is identified by its ID. Valid processor ID values are 0 through 7.

Memory Access Features

The external multiprocessor address is divided into two fields, as illustrated in Table 2-3 on page 2-6.

Table 2-3. External Multiprocessor Space

Bits	Name or Value	Definition
ADDR21–0	Address	Internal address space, as described in “Internal Address Space” on page 2-6
ADDR31–22	PRID	Processor ID—determine the target processor. 1NNN defines one of eight possible processors, where <i>NNN</i> is the target TigerSHARC processor ID. 0111 – Broadcast—access (which can be only write) is to all TigerSHARC processors in the cluster. 0000 – Defines the memory bank as internal

 The TigerSHARC processor’s own internal space (including UREGs) can be accessed via the multiprocessing space (including broadcast space) for write transactions only. There are no interlocks. Because this is performed through the external bus it should only be used in special cases where data must pass through the TigerSHARC processor bus interface.

Internal Address Space

The internal address space corresponds to that processor’s own internal address space, UREGS. The internal address space is described by bits ADDR21–0 of multiprocessing or internal space. The internal address is divided into three fields, as illustrated in Table 2-4 on page 2-7.

Internal space is the space for transactions within the TigerSHARC processor and access to this memory space is not reflected on the cluster bus. Internal address space is used to access the internal memory blocks, or any of the Universal registers (Uregs). Universal registers are internal registers that are mapped to the TigerSHARC processor memory map. Most software accessible registers are Uregs.

Table 2-4. Internal Address Space

Bits	Name or Value	Definition
ADDR18–0	Address	Address
ADDR21–19	ISPACE	Internal Space—determines the internal space. 000 – Block 0 001 – Block 1 010 – Block 2 011 – Internal Registers (Uregs) 1XX – Reserved
ADDR31–22	PRID	Internal space if 000 00

Access to Uregs (ISPACE = 011) is only for multiprocessing space. Internal access to registers cannot be memory mapped—in load/store instructions, for example, the address cannot point to a register through the internal space. The DMA also cannot access a register directly, although there is an exception—link receive DMA channels may write to other link transmit registers.

Internal Memory Access

In addition to the direct accesses (normal - one 32-bit word, long - two 32-bit words, and quad - four 32-bit words), several other efficient methods are available. These include Broadcast Write, Merged Distribution, and Broadcast Distribution. Broadcast Write is an external write to other DSPs in a multiprocessor cluster. Merged and Broadcast Distribution are internal access methods. For additional information on memory access methods, see “IALU” in the *ADSP-TS101 TigerSHARC Processor Programming Reference*. “Merged Access” on page 2-11 also provides additional information regarding these access modes.

Memory Access Features

Broadcast Write

In Broadcast Write, a TigerSHARC processor or host writes the *same* data to different TigerSHARC processors via multiprocessor memory space. One TigerSHARC processor writes to the Broadcast memory space using the external bus to distribute the same stream of data to all processors. A common application would be when several processors perform different algorithms on the same stream of data.

The number of source locations and destination registers listed in the instruction determines whether broadcast or merged distribution is used. Distribution is commonly used in a FIR filter application, where data is loaded in quads then re-arranged into the groups needed. This requires a Data Alignment Buffer, described in the next section.

Merged Distribution

In Merged Distribution, one instruction loads *different* data into two compute blocks in the same TigerSHARC processor. Instruction syntax determines how the data is distributed. For example, xyRq identifies two quad-word Uregs, one in processing element X and one in processing element Y. When the prefixes are absent the assembler assumes the order xy, reverse the order of the processing elements by reversing the prefix (yx).

For long-word access, the processor distributes the MSW to processing element X, and the LSW to element Y. For the quad-word access, two MSWs are distributed to processing element X, and two LSWs are distributed to element Y.

Broadcast Distribution

In Broadcast Distribution, one instruction loads the *same* data into two compute blocks in the *same* TigerSHARC processor. Broadcast distribution works with normal, long, or quad data accesses.

Data Alignment Buffer

Normally load instructions must be aligned to their data size so that quad-words are loaded from a quad-aligned address. But some applications need to access non-aligned quad-words in memory. This can be done by using the Data Alignment Buffer (DAB) instructions. Failure to use a DAB instruction in accessing a non-aligned word results in an error.

The DAB is a quad-word FIFO in which the load from memory is aligned, but the transfer to the processing elements register file is non-aligned. Part of the data comes from a previous load from memory, and part comes from the current load from memory. Thus at the start of each new DAB access, the application must issue two consecutive accesses. The first is a dummy access to clear the DAB of old data, the second access delivers valid data. This is discussed on page 6-23 of the *ADSP-TS101 TigerSHARC Processor Programming Reference*.

Register Access Features

Registers can be accessed in two ways—either by the core with a register access instruction, or as a memory-mapped register by a different TigerSHARC processor or Host that acts as a master. For example, two fields in the memory mapped address, Ureg group and Ureg, can be used to access the register via a register access instruction.

Register Space

The register space is composed of 64 register groups with up to 32 registers in each group. Register groups are defined in the range 0x3F-0 (63-0), where groups 0x1F-0 are accessible by all transfer instructions (load immediate, move register, load and store), and groups 0x3F-0x20 are accessible only by move register instructions and direct accesses of other masters.

Register Access Features

Some of the registers do not use all 32 bits. The unimplemented bits are reserved, and should not be used. When writing to such a register, they must be set to zero. When reading such a register, the unimplemented bits may be of any value.

The register groups are:

- 0x00 – 0x09 Compute block register file different aliases
- 0x0C – 0x0F Integer ALU registers
- 0x1A, 0x38 – 0x39 Interrupt and sequencer registers
- 0x30 – 0x37, 3B Branch Target Buffer (BTB) registers
- 0x1B, 0x3D, 0x3E, 0x3F Debug logic (0x3F reserved)
- 0x24, 3A EP Control/Status registers
- 0x20 – 0x23 DMA registers
- 0x25 – 0x27 Link registers
- Others – Reserved. These registers must not be accessed by applications since they could cause unexpected behavior by the TigerSHARC processor.

The register groups are described in the following sections.


Compute Block Register Files

The *compute block register file* is a 32-word register. There are two such register files, one in each of the compute blocks (X and Y). The register file is aliased in several groups, some of which are applicable for data transfer instructions only. Some groups point to compute block X and some to block Y; those that point to both compute blocks in parallel actually point to the same register.

Merged Access

One type of access to both X and Y register files is the *merged access*. Merged access provides efficient data load and store for SIMD applications, accessing both compute blocks in one cycle.

Merged access splits the data between the two compute blocks. If the data transfer is specified as a quad-word, two words are transferred to CBX and two are transferred to CBY. If the access is long, one word is transferred to CBX, and another is transferred to CBY.

 The transaction of a normal word to a merged group is illegal.

Broadcast Transfer

For broadcast transfer, the same data is written to CBX and CBY simultaneously. Broadcast duplicates the data from memory to both compute blocks and can only be used for write-to-compute registers. For more information refer to the *ADSP-TS101 TigerSHARC Processor Programming Reference*.

Register Access Features

The Compute Block register types of accesses are listed in Table 2-5.

Table 2-5. Compute Block RF Groups

Address	Group Definition	Ureg Group
0x180000–0x18001F	Compute block X (CBX) register file regular access	0b000000
Not accessible ¹	Compute block X (CBX) register file regular access alternative	0b000001
0x180040–0x18005F	CBY register file regular access	0b000010
Not accessible ¹	CBY register file regular access alternative	0b000011
0x180080–0x18009F	CBX and CBY merged	0b000100
Not accessible ¹	CBX and CBY merged alternative	0b000101
0x1800C0–0x1800DF	CBY and CBX merged	0b000110
Not accessible ¹	CBY and CBX merged alternative	0b000111
0x180100–0x18011F	CBX and CBY broadcast	0b001000
Not accessible ¹	CBX and CBY broadcast alternative	0b001001

¹ Alternative access is only for load/store instruction—DAB, circular buffer.

Unmapped Compute Block Registers

There are several registers in the compute block units that are not Universal registers (Uregs), in that they are not accessed in the same way that Uregs are accessed. These registers are accessed by regular compute block instructions that transfer the data between the unmapped registers and the general-purpose compute block Uregs.

Global Registers—XSTAT/YSTAT Compute Block Status Registers

XSTAT and YSTAT are 32-bit compute block status registers that record the state of the compute block status flags. Every flag is updated when an instruction belonging to its computation unit is completed. For more information regarding the X/YSTAT registers, see “Compute Block Registers” in the *ADSP-TS101 TigerSHARC Processor Programming Reference*.

ALU Registers

The Parallel Results registers (PRO and PR1) are two 32-bit registers used for sideways sum instructions. For more information regarding the PR_x registers, see “ALU” in the *ADSP-TS101 TigerSHARC Processor Programming Reference*.

Multiplier Registers

The Multiplier Results registers (MR3-0 and MR4) are used as accumulators for the different types of fixed-point Multiply-Accumulate instructions. For more information regarding the MR_x registers, see “Multiplier” in the *ADSP-TS101 TigerSHARC Processor Programming Reference*. These registers are accessed by different multiplier instructions.

Shifter Registers

The Bit FIFO Overflow register (BFOTMP) is a 64-bit register used in the PUTBITS instruction. For more information regarding the BFOTMP register, see the PUTBITS instruction in the *ADSP-TS101 TigerSHARC Processor Programming Reference*.

Communications Logic Unit (CLU) Registers

The CLU instructions use 16 Trellis (TR15-0) data registers. Each register is 32 bits wide.

The Trellis history registers (THR1-0) are 32-bit registers.

Register Access Features

This register pair keeps the history of ACS selection decisions. On each ACS instruction execution $THR1-0$ are shifted right, and the selection vector (one bit for each ACS decision) is shifted into $THR1-0$ MSBs (most significant bits). The shift is by four in 32-bit operations, and by eight in 16-bit operations.

This register is also used for bit representation of spreading code and scrambling code in `DESPREAD` functions.

IALU Registers

Each IALU has two Ureg groups. The first group is the general-purpose register file, which includes 32 normal-word registers each. The second group is the circular buffer register file which is used to specify parameters for circular buffering. The IALU register groups are listed in Table 2-6 on page 2-14.

The circular buffer register files contain these registers:

- 3–0 circular buffer base registers $JB3-0$ and $KB3-0$
- 7–4 circular buffer length registers $JL3-0$ and $KL3-0$
- 31–8 Reserved

Table 2-6. IALU RF Groups

Address	Group Definition	Group
0x180180 – 0x18019F	J-IALU register file	0x0C
0x1801A0 – 0x1801BF	K-IALU register file	0x0D
0x1801C0 – 0x1801DF	J-IALU circular buffer register file	0x0E
0x1801E0 – 0x1801FF	K-IALU circular buffer register file	0x0F

Register J31 – JSTAT

The JSTAT status register records the state of the c flags and is updated as a result of various IALU instructions. When being written to, the JSTAT register can be referred to as J31. When used as an operand in an IALU arithmetic, logical, or function operation — J31 and K31 are treated as zero. See the IALU chapter of the *ADSP-TS101 TigerSHARC Processor Programming Reference*.

Table 2-7. JSTAT Register Bit Descriptions

Bits	Name	Definition
0	JZ	J-IALU zero
1	JN	J-IALU negative
2	JV	J-IALU overflow
3	JC	J-IALU carry
31 – 4	Reserved	Reserved

Register K31 – KSTAT

The KSTAT status register records the state of the K-IALU flags and is identical to JSTAT except for the bits names. As a Ureg, KSTAT is mapped as register 31 (K31) in the K-IALU register group.

Sequencer Register Groups

All sequencer registers (including the BTB address range) are accessible by single-word access only.

The 32-bit register group 0x1A is dedicated to sequencer registers, interrupt control, and status registers. See Table 2-8.

Register Access Features

For more information regarding the sequencer registers, see “Program Sequencer” in the *ADSP-TS101 TigerSHARC Processor Programming Reference*.

Sequencer Control Register – SQCTL

The sequencer is controlled and configured by writing to the SQCTL register, which is set to 0x0100 after reset.

Sequencer Control Register Set Bits – SQCTLST

The SQCTLST bit is an alias used to write to SQCTL. When writing to this address, the data written into SQCTL is the OR of the old register value and the new data written into SQCTLST. A ‘1’ in any bit of the written data sets the corresponding bit in SQCTL, while a ‘0’ in written data does not change the bit value.

Sequencer Control Register Clear Bits – SQCTLCL

The SQCTLCL bit is an alias used to write to SQCTL. When writing to this address, the data written into SQCTL is the AND of the old register value and the new data written into SQCTLCL. This way a ‘0’ in any bit of the written data clears the corresponding bit in SQCTL, while a ‘1’ in written data does not change the bit value.

Sequencer Status Register – SQSTAT

This is a read-only register that holds information about the current status of the sequencer. The initial value of this register after reset is 0xFF00.

SFREG Register

Static flags are used as static copies of conditions. When the programmer wishes to keep a condition value after another instruction changes its value, it can be copied into the SFREG and used later as a condition. All

Table 2-8. Sequencer Registers

Single Word	Register Type	Direct Memory Address	Remarks
CJMP	Address branch (see CJUMP Instruction in the <i>ADSP-TS101 TigerSHARC Processor Programming Reference</i>)	0x180340	
RETI	Address for return from interrupt (see the <i>ADSP-TS101 TigerSHARC Processor Programming Reference</i>)	0x180342	
RETIB	Alias of RETI for nesting interrupts (see “Interrupts” on page 4-1)		
RETS	Return address for exception (see “Exceptions” on page 4-23)	0x180344	
DBGE	Return address for emulation trap (see “Exceptions” on page 4-23)	0x180345	
ILATSTL	ILAT low set—used to write to ILAT	0x180346	Write only register
ILATSTH	ILAT high set—used to write to ILAT	0x180347	Write only register
LC0	Loop counter #0	0x180348	
LC1	Loop counter #1	0x180349	
ILATL	ILAT low—used to read ILAT (see “ILAT Register” on page 4-11)	0x18034A	Read only register Reset value 0x0
ILATH	ILAT high—used to read ILAT (see “ILAT Register” on page 4-11)	0x18034B	Read only register Reset value 0x0
IMASKL	IMASK low (see “IMASK Register” on page 4-12)	0x18034C	Reset value 0xE3C3C000
IMASKH	IMASK high (see “IMASK Register” on page 4-12)	0x18034D	Reset value 0x90010061 or 0x90011E61 ¹
PMASKL	PMASK low (see “PMASK Register” on page 4-13)	0x18034E	Read only Reset value 0x0

Register Access Features

Table 2-8. Sequencer Registers (Cont'd)

Single Word	Register Type	Direct Memory Address	Remarks
PMASKH	PMASK high (see “PMASK Register” on page 4-13)	0x18034F	Read only Reset value 0x0
TIMER0L	Timer 0 low (see “Timers” on page 4-4)	0x180350	Read only
TIMER0H	Timer 0 high (see “Timers” on page 4-4)	0x180351	Read only
TIMER1L	Timer 1 low (see “Timers” on page 4-4)	0x180352	Read only
TIMER1H	Timer 1 high (see “Timers” on page 4-4)	0x180353	Read only
TMRIN0L	Timer 0 initial value low (see “Timers” on page 4-4)	0x180354	
TMRIN0H	Timer 0 initial value high (see “Timers” on page 4-4)	0x180355	
TMRIN1L	Timer 1 initial value low (see “Timers” on page 4-4)	0x180356	
TMRIN1H	Timer 1 initial value high (see “Timers” on page 4-4)	0x180357	
SQCTL	Sequencer control register	0x180358	Reset value 0x100 or 0xF01001
SQCTLST	Sequencer control register set bits	0x180359	
SQCTLCL	Sequencer control register clear bits	0x18035A	
SQSTAT	Sequencer status register	0x18035B	Read only, Reset value 0xFF04
SFREG	Static condition flag register	0x18035C	
ILATCLL	ILAT low clear—used to clear ILAT	0x18035E	Write only register
ILATCLH	ILAT high clear—used to clear ILAT	0x18034F	Write only register

1 According to IRQEN strap.

static condition flags are grouped into the SFREG register. (See the *ADSP-TS101 TigerSHARC Processor Programming Reference* for information about how this register is updated and used).

ILAT Registers

The ILAT register is a single 64-bit register accessed as two 32-bit registers ILATH and ILATL. Each bit is dedicated to an interrupt. When an interrupt occurs, the corresponding bit is set. The interrupt bits order is set according to the interrupt priority—bit 0 having lowest priority.

The ILAT register can be written through the set (ILATSTL or ILATSTH) and clear (ILATCLL or ILATCLH) addresses. Writing to the set addresses updates the ILAT register to the OR of the old and written values. As a result, every set bit in the written data is set the corresponding bit in the ILAT register. Writing to the clear address will AND the data written with the old data of the ILAT register. In this case, a zero bit in the input data clears the corresponding bit in ILAT, while a set bit keeps it unchanged. The consequences and restrictions of these accesses are detailed in “ILAT Register” on page 4-11.

IMASK Register

The IMASK register is a single 64-bit register accessed as two 32-bit registers, IMASKL and IMASKH.

The initial value of IMASK after reset is:

0x9001006 1E3C3C000 – Normal reset

0x90011E6 1E3C3C000 – Reset with $\overline{\text{TRQ}}$ enable strap active

PMASK Register

The PMASK register is a single 64-bit register that is accessed as two 32-bit registers, PMASKL and PMASKH. The initial value of PMASK after reset is 0x0.

Register Access Features

Interrupt Vector Table Register Groups

The 32-bit register groups 0x38 and 0x39 are dedicated to interrupt control—see Table 2-9, Table 2-10 and “Interrupt Vector Table” on page 4-2.

Table 2-9. Interrupt Vector Table (IVT) for Group 0x38

Register Name	Register Type	Direct Memory Address	Reset Value
IVTIMER0LP	Timer #0 low priority	0x180702	
IVTIMER1LP	Timer #1 low priority	0x180703	
IVLINK0	Link #0 Reg	0x180706	
IVLINK1	Link #1 Reg	0x180707	
IVLINK2	Link #2 Reg	0x180708	
IVLINK3	Link #3 Reg	0x180709	
IVDMA0	DMA #0 Reg	0x18070E	0x0
IVDMA1	DMA #1 Reg	0x18070F	0x0
IVDMA2	DMA #2 Reg	0x180710	0x0
IVDMA3	DMA #3 Reg	0x180711	0x0
IVDMA4	DMA #4 Reg	0x180716	0x0
IVDMA5	DMA #5 Reg	0x180717	0x0
IVDMA6	DMA #6 Reg	0x180718	0x0
IVDMA7	DMA #7 Reg	0x180719	0x0
IVDMA8	DMA #8 Reg	0x18071D	0x0
IVDMA9	DMA #9 Reg	0x18071E	0x0
IVDMA10	DMA #10 Reg	0x18071F	0x0

Table 2-10. Interrupt Vector Table (IVT) for Group 0x39

Register Name	Register Type	Direct Memory Address	Reset Value
IVDMA11	DMA #11 Reg	0x180720	0x0
IVDMA12	DMA #12 Reg	0x180725	0x0
IVDMA13	DMA #13 Reg	0x180726	0x0
IVIRQ0	IRQ0 Reg pin	0x180729	0x1000 0000
IVIRQ1	IRQ1 Reg pin	0x18072A	0x0800 0000
IVIRQ2	IRQ2 Reg pin	0x18072B	0x0C0 0000
IVIRQ3	IRQ3 Reg pin	0x18072C	0x0
VIRPT	VIRPT (vector Reg)	0x180730	
IVBUSLK	Bus lock vector	0x180732	
IVTIMER0HP	Timer 0 high priority	0x180734	
IVTIMER1HP	Timer 1 high priority	0x180735	
IVHW	Hardware error	0x180739	
IVSW	Software exception	0x18073E	

Debug Register Groups

The debug registers use groups 0x3D, 0x3E, and 0x1B. These register groups are described in Table 2-11 on page 2-22 through Table 2-13 on page 2-23. The debug registers can only be accessed as single-word registers.

In emulation mode the debug registers can be accessed only by move, register-to-register, or immediate data load instructions. These registers cannot be loaded from or stored to memory directly.

Register Access Features

Table 2-11. Debug Register Groups for 0x3D Register Numbers

Register Name	Register Type	Direct Memory Address	Remarks
WPOCTL	Watchpoint 0 control	0x1807A0	Reset value 0x0
WP1CTL	Watchpoint 1 control	0x1807A1	Reset value 0x0
WP2CTL	Watchpoint 2 control	0x1807A2	Reset value 0x0
WPOSTAT	Watchpoint 0 status	0x1807A4	Read only, reset value 0x0XXXXX
WP1STAT	Watchpoint 1 status	0x1807A5	Read only, reset value 0x0XXXXX
WP2STAT	Watchpoint 2 status	0x1807A6	Read only, reset value 0x0XXXXX
WP0L	Watchpoint 0 low address	0x1807A8	
WP0H	Watchpoint 0 high address	0x1807A9	
WP1L	Watchpoint 1 low address	0x1807AA	
WP1H	Watchpoint 1 high address	0x1807AB	
WP2L	Watchpoint 2 low address	0x1807AC	
WP2H	Watchpoint 2 high address	0x1807AD	

Table 2-12. Debug Register Groups for 0x1B Register Numbers

Register Name	Register Type	Direct Memory Address	Remarks
PRFM	Performance monitor mask	0x180363	Reset value 0x0
CCNT0	Cycle counter low	0x180364	Reset value 0x01
CCNT1	Cycle counter high	0x180365	Reset value 0x01
PRFCNT	Performance monitor counter	0x180366	Reset value 0x0
TRCBMASK	Trace buffer mask	0x180370	Read only, reset value 0xFFFFF00
TRCBPTR	Trace buffer pointer	0x180378	Read only, reset value 0x0

Table 2-13. Debug Register Groups for 0x3E Register Numbers

Register Name	Register Type	Direct Memory Address	Remarks
TRCB0	Trace buffer 0	0x1807C0	Read only, reset value 0x0
TRCB1	Trace buffer 1	0x1807C1	Read only, reset value 0x0
TRCB2	Trace buffer 2	0x1807C2	Read only, reset value 0x0
TRCB3	Trace buffer 3	0x1807C3	Read only, reset value 0x0
TRCB4	Trace buffer 4	0x1807C4	Read only, reset value 0x0
TRCB5	Trace buffer 5	0x1807C5	Read only, reset value 0x0
TRCB6	Trace buffer 6	0x1807C6	Read only, reset value 0x0
TRCB7	Trace buffer 7	0x1807C7	Read only, reset value 0x0

Performance Monitor Counter – PRFCNT

The purpose of the performance counter is to register the cycle where monitored conditions occur. The monitored condition is identified by the performance monitor mask register (PRFM). The register is cleared after reset.

Performance Monitor Mask – PRFM

The performance monitor identifies which of the following events is counted by the performance monitor counter (PRFCNT). Bits30–0 indicate which events are monitored and Bit31 indicates if the events in the same cycle are summed or ORed. The entire register is cleared after reset. The bit descriptions for this register are shown in Figure 2-2 on page 2-24 and Figure 2-3 on page 2-25.

Register Access Features

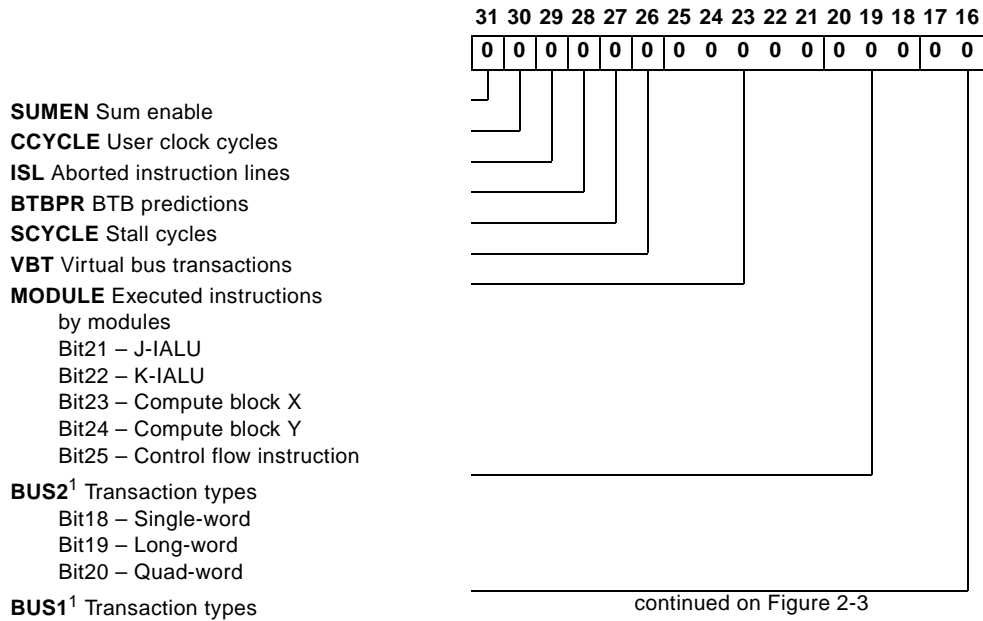


Figure 2-2. PRFM (Upper) Register Bit Descriptions

- 1 Monitoring is performed on transactions driven on bus n (n is 0, 1, or 2), including virtual bus transactions

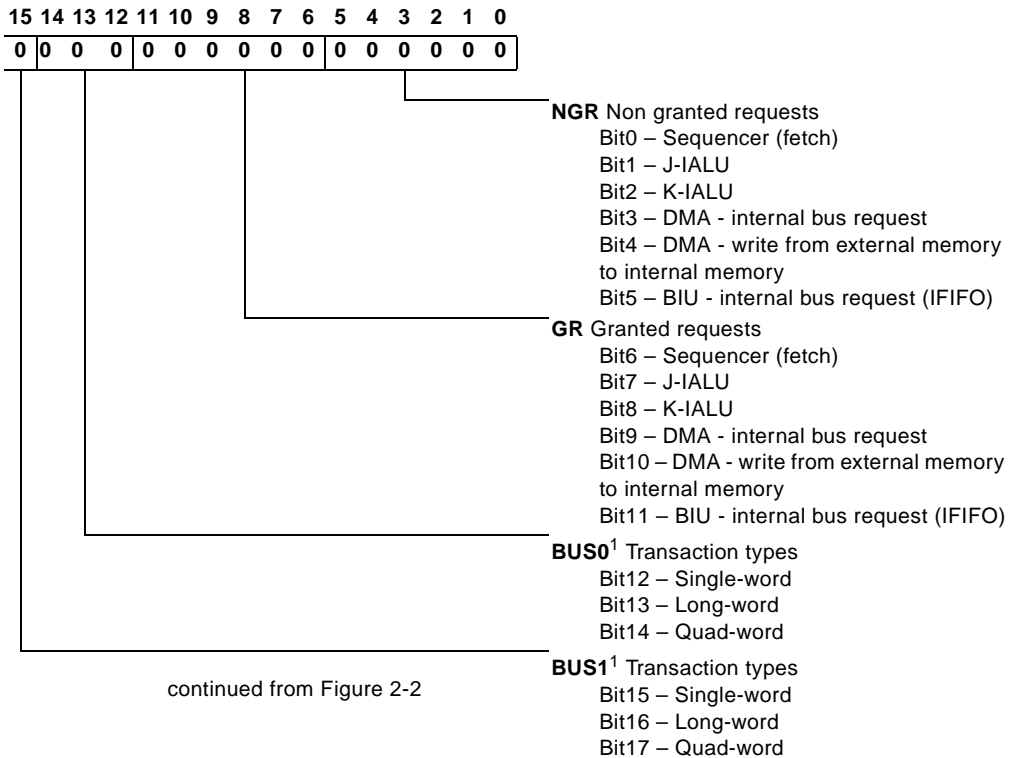


Figure 2-3. PRFM (Lower) Register Bit Descriptions

1 Monitoring is performed on transactions driven on bus n (n is 0, 1, or 2), including virtual bus transactions

Watchpoint Control – WP0CTL, WP1CTL and WP2CTL

Each of the three watchpoints has a control register used to define its operation. After reset, the operating mode is in “watchpoint disabled” (state 00). The bit descriptions for this register are shown in Figure 2-4 on page 2-26 and Figure 2-5 on page 2-27.

Register Access Features

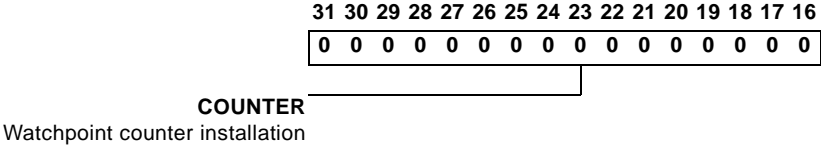


Figure 2-4. WPxCTL (Upper) Register Bit Descriptions

Memory and Register Map

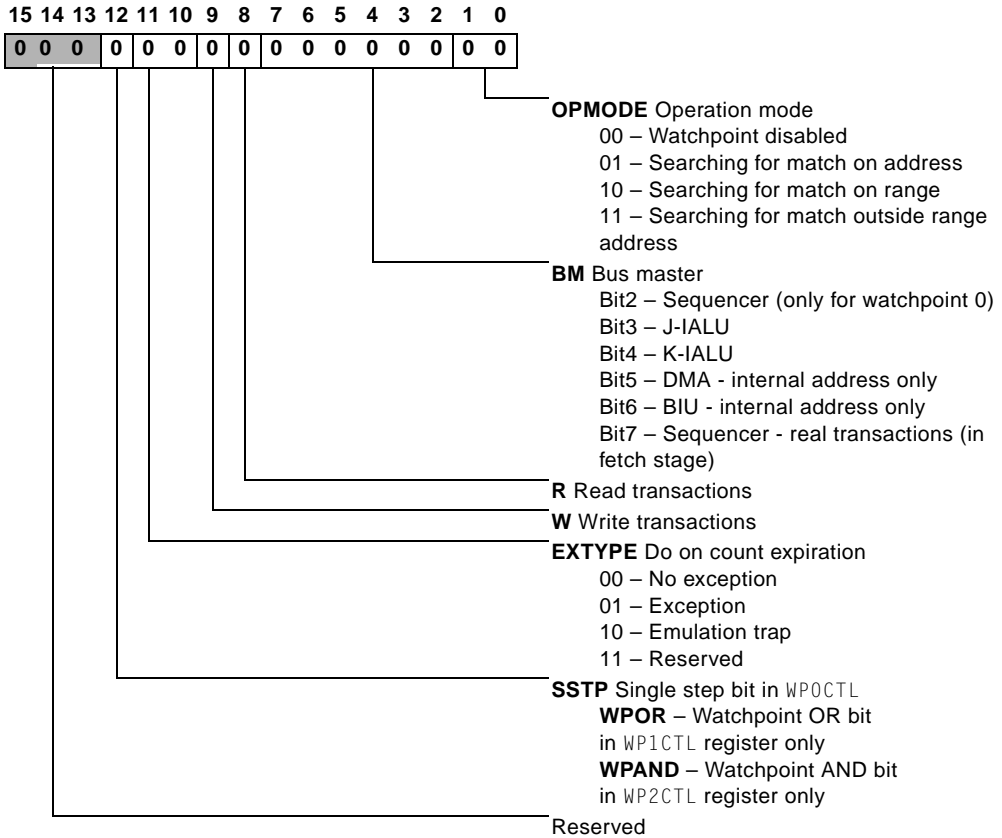


Figure 2-5. WPxCTL (Lower) Register Bit Descriptions

Watchpoint Status – WP0STAT, WP1STAT and WP2STAT

Each of the three watchpoints has a status register used to indicate its operation. After reset, the EX field is in “watchpoint disabled” (state 00). The bit descriptions for this register are shown in Figure 2-6 on page 2-28 and Figure 2-7 on page 2-29.

Register Access Features

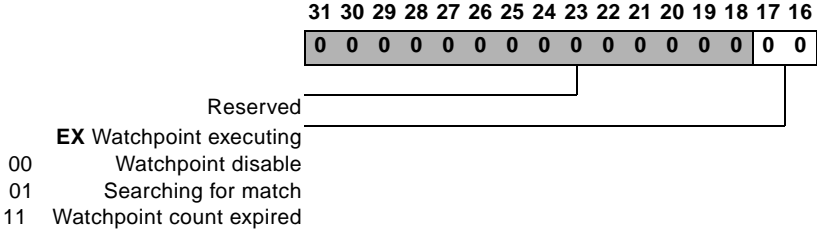


Figure 2-6. WPxSTAT (Upper) Register Bit Descriptions

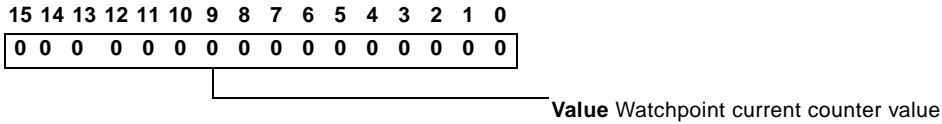


Figure 2-7. WPxSTAT (Lower) Register Bit Descriptions

Cycle Counters – CCNT0 and CCNT1

The cycle counter is 64 bits long. The two Uregs that make up the cycle counter are CCNT0 and CCNT1. When JTAG is reset, CCNT is cleared. These two registers cannot be accessed with a long-word access; they can only be accessed in two single accesses. These registers can be written in emulator or supervisor modes only, although they are readable in all modes.

Trace Buffer – TRCB0 to TRCB7 and TRCBPTR

Each time the TigerSHARC processor executes a non-sequential fetch, the PC of the non-sequentially fetched instruction is written into one of the trace buffer registers. The first write is into trace buffer #0; the second to #1, and so on in a circular manner. The trace buffer pointer (3 bits) identifies the last written trace buffer.

After reset, the trace buffer registers (TRCB) and pointer (TRCBPTR) are all set to zero. All the trace buffer registers are read by software only.

For more information, see “Instruction Address Trace Buffer (TBUF)” on page 9-9.

Watchpoint Address Pointers – WP0L, WP1L, WP2L, WP0H, WP1H and WP2H

The watchpoint address pointers are 32-bit pointers defining the address, or address range, of the watchpoints. Their value after reset is undefined.

Register Access Features

External Port Registers

The external port (EP) includes:

- Bus Control/Status registers (BIU)
- External Port Configuration and Status registers
- External Port DMA registers
- AutoDMA registers
- Link Port Receive/Transmit DMA registers
- DMA Control and Status registers
- Link Port Control and Status registers
- Link Port Buffer registers

Bus Control/Status (BIU) Register Group

The Bus Control/Status registers are defined in the following sections.

SYSTAT/SYSTATCL Register

The SYSTAT register is a read-only register that indicates the system/bus status. The SYSTAT register can be read using the name SYSTAT corresponding to address 0x6 or SYSTATCL corresponding to address 0x7. The address determines the value of the error bits after the read. The bit descriptions for this register are shown in Figure 2-8 on page 2-32 and Figure 2-9 on page 2-33.

- SYSTAT – no change in register contents
- SYSTATCL – error Bits19–16 are cleared after the read

Register Access Features

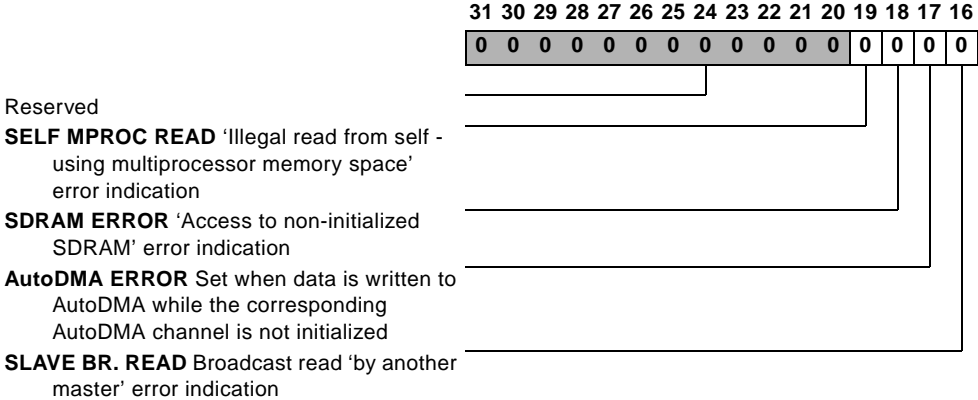


Figure 2-8. SYSTAT/SYSTATCL (Upper) Register Bit Descriptions

Memory and Register Map

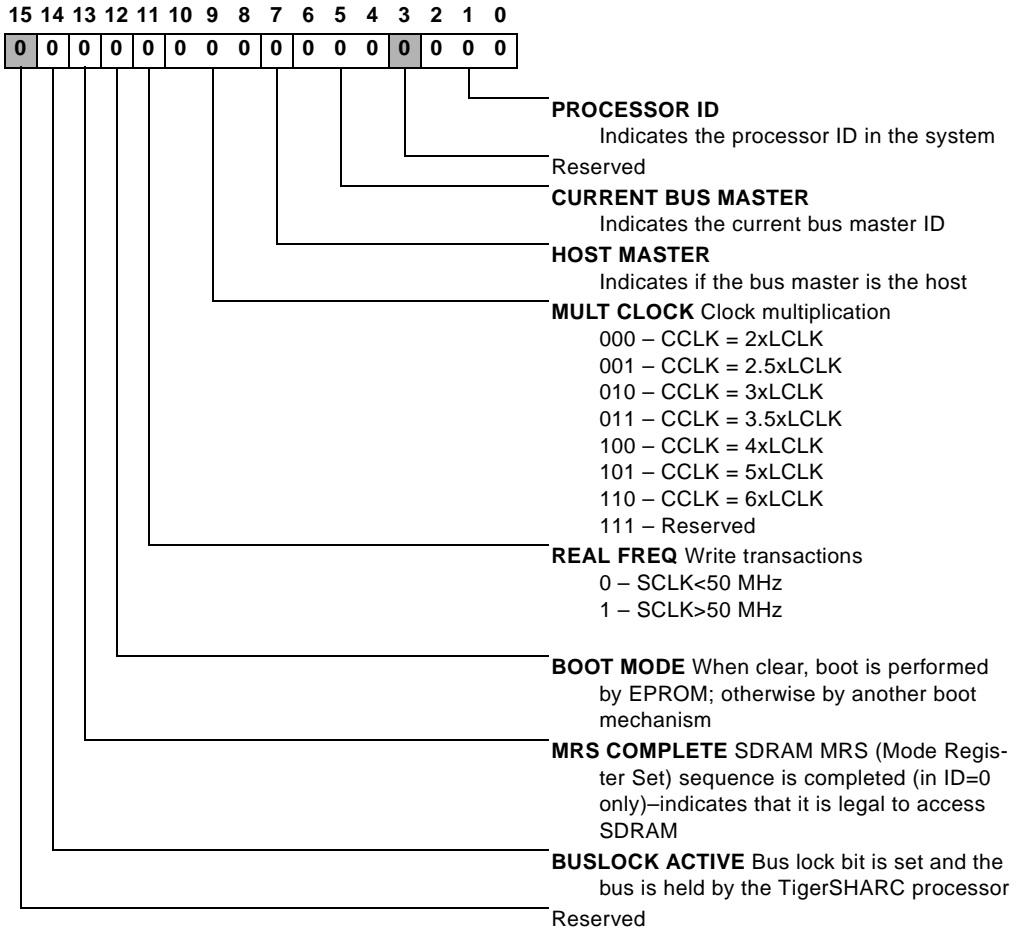


Figure 2-9. SYSTAT/STATCL (Lower) Register Bit Descriptions

Register Access Features

SYSCON Register (DMA 0x180480)

The SYSCON register defines bus control configuration. It can be written only once after hardware reset and cannot be changed during system operation, subsequent writes to SYSCON are ignored. The initial value of SYSCON after reset is 0x000279E7, which defines slow protocol with three wait states and a 32-bit bus width for all buses. The bit descriptions for this register are shown in Figure 2-10 on page 2-34 and Figure 2-11 on page 2-35.

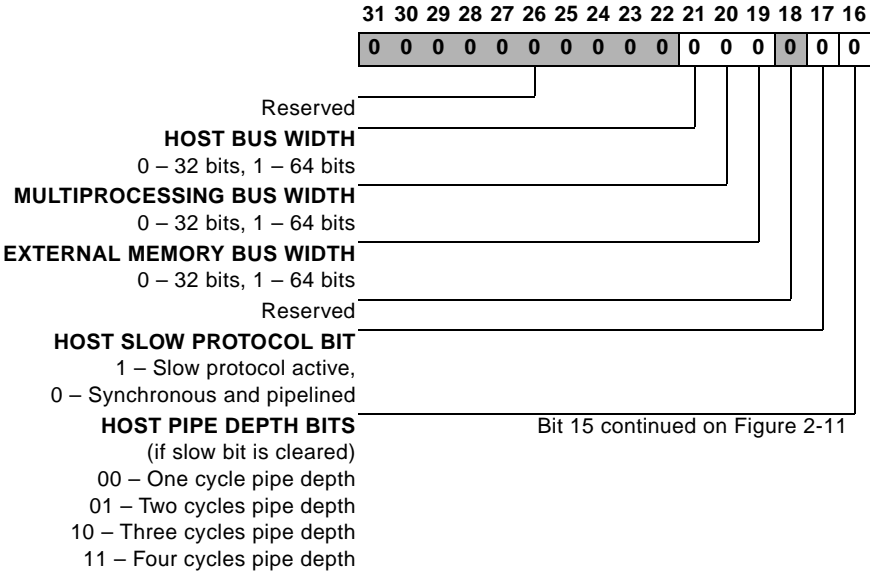


Figure 2-10. SYSCON (Upper) Register Bit Descriptions

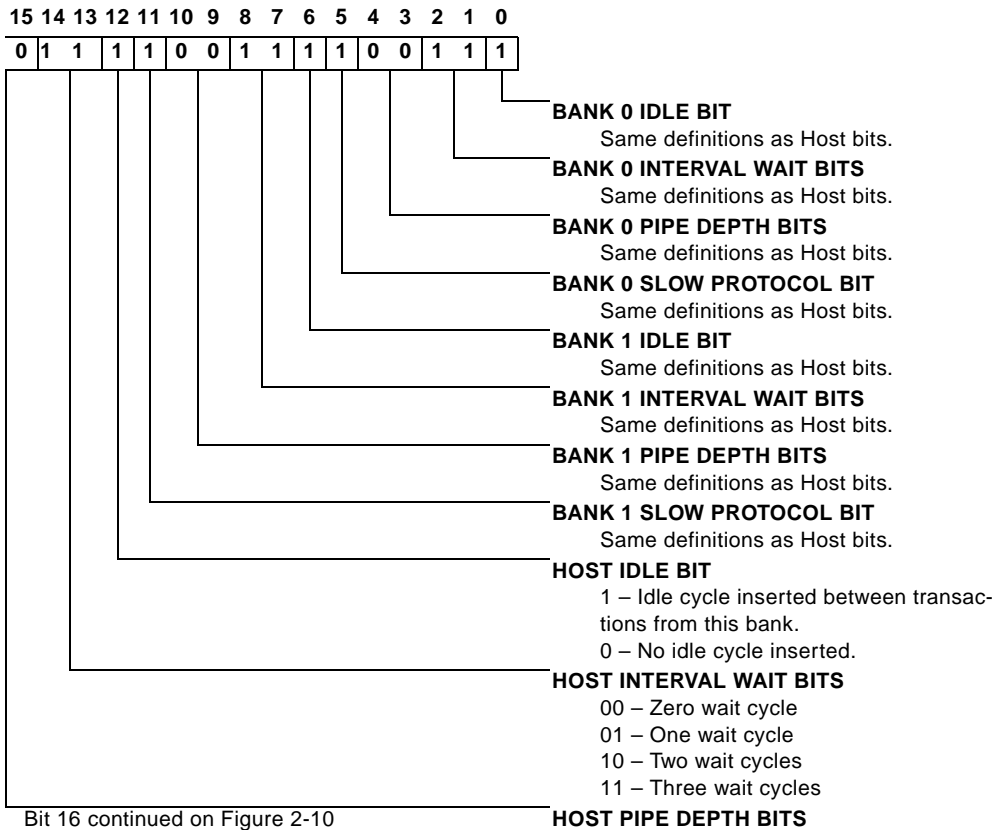


Figure 2-11. SYSCON (Lower) Register Bit Descriptions

Register Access Features

SDRCON (SDRAM Configuration) (DMA 0x180484)

The SDRCON register defines SDRAM configuration. The SDRCON register can be written only once after reset and cannot be changed during system operation. The initial value of the SDRCON register after reset is zero, meaning that the SDRAM is disabled.

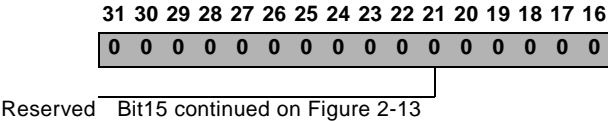


Figure 2-12. SDRCON (Upper) Register Bit Descriptions

The bit descriptions for this register are shown in Figure 2-12 on page 2-36 and Figure 2-13 on page 2-37.

Memory and Register Map

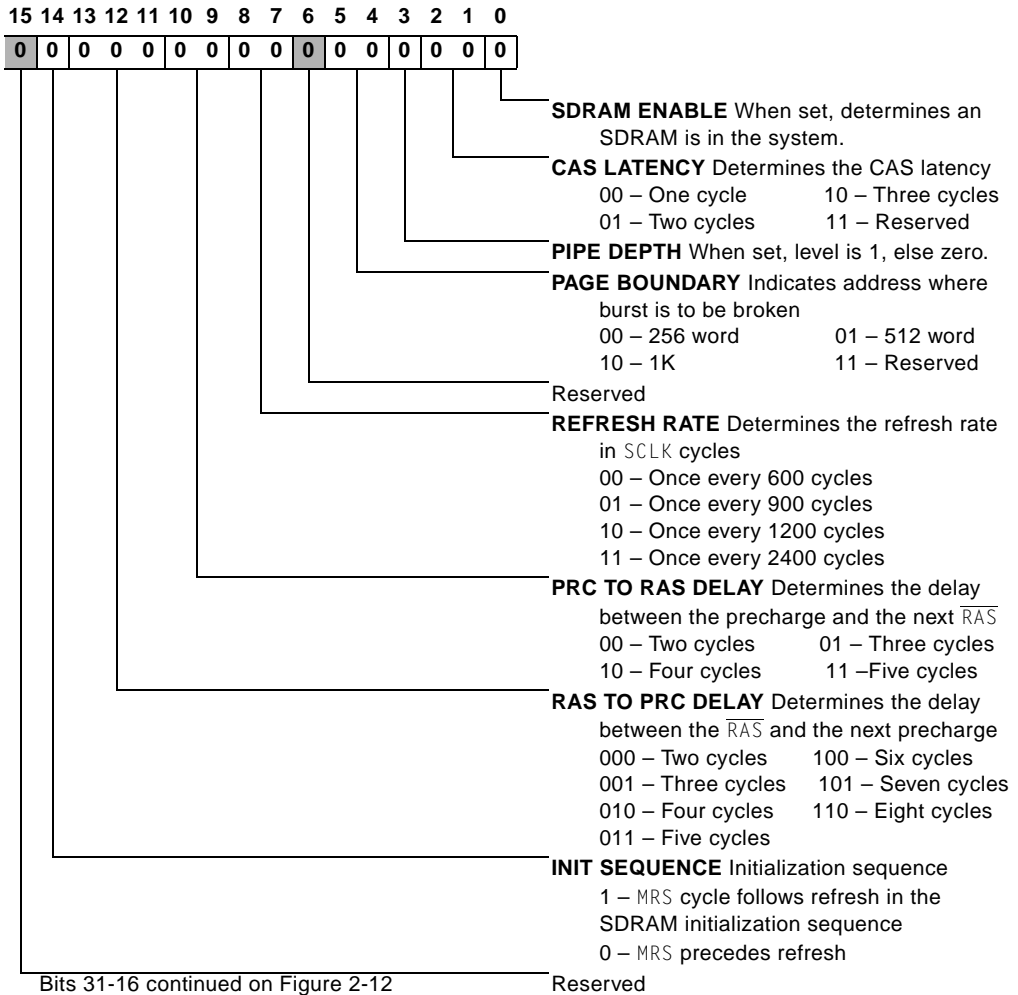


Figure 2-13. SDRCON (Lower) Register Bit Descriptions

Register Access Features

BUSLK System Control

The BUSLK register defines the bus lock status. The initial value of the BUSLK register after reset is zero.

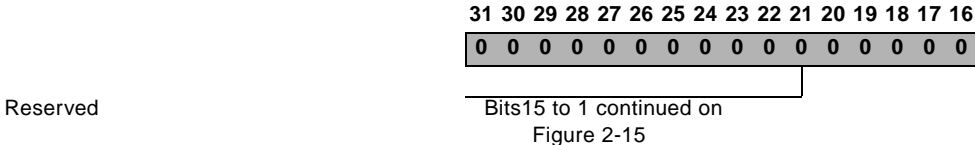


Figure 2-14. BUSLK (Upper) Register Bit Descriptions

The bit descriptions for this register are shown in Figure 2-14 on page 2-38 and Figure 2-15 on page 2-39.

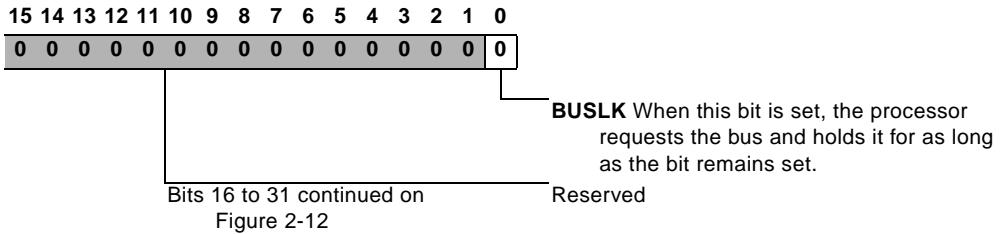


Figure 2-15. BUSLK (Lower) Register Bit Descriptions

BMAX Current Value

This address is read-only and returns the **BMAX** counter value when read.

BMAX Register

The **BMAX** register is set with the maximum number of internal clock cycles (**CLK**) for which the TigerSHARC processor is allowed to retain mastership on the external bus. When reading this address, the returned value is the value written to the **BMAX** register, not the current cycle count. After reset, the **BMAX** register value is 0xFFFF.

The **BMAX** register is automatically loaded after reset and begins to decrement when the TigerSHARC processor attains bus mastership. THE countdown cannot be disabled. See “Bus Fairness — **BMAX**” on page 5-47 for more information. The bit descriptions for this register are shown in Figure 2-16 on page 2-40 and Figure 2-17 on page 2-41.

Register Access Features

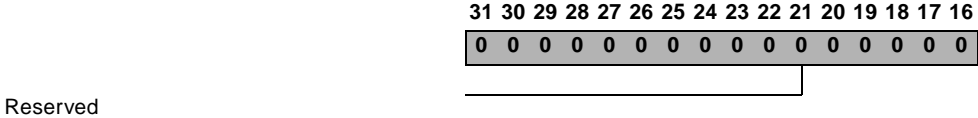


Figure 2-16. BMAX (Upper) Register Bit Descriptions

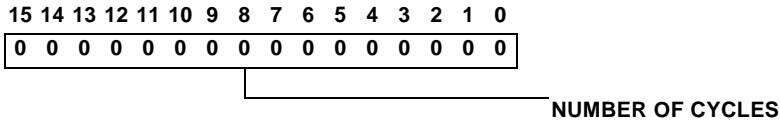


Figure 2-17. BMAX (Lower) Register Bit Descriptions

External Port Configuration and Status Registers

Table 2-14. External Port Configuration and Status Registers

Register Quad	BIU Registers	Direct Memory Address	Reset Value
SYSCON ¹	System configuration register (see “SYSCON Programming” on page 5-11)	0x180480	0x279E7
BUSLK	System control register	0x180483	0x0
SDRCON ¹	SDRAM configuration register (see “SDRAM Programming” on page 6-19)	0x180484	0x0
SYSTAT	System status register	0x180486	Read only
SYSTATCL	System status register destructive address (clear error)	0x180487	Read only
BMAX	Maximum cycle count for bus fairness (see “Bus Fairness — BMAX” on page 5-47)	0x18048C	0xFFFF
BMAXC	Current count on BMAX	0x18048D	0xFFFF Read only

¹ Can be written only once after hardware reset. After first write, it becomes a read-only register.

DMA Registers

The DMA register groups are described in Table 2-15 on page 2-43 to Table 2-19 on page 2-46. The DMA TCB registers are only accessible via quad-word accesses.

Register Access Features

The Control register shown in Table 2-19 on page 2-46 has three addresses:

- DCNT Regular address – read and write

The initial value of the DCNT after reset is 0x0.

- DCNTST Set bits – write. When writing to this address, the register's old value is OR'ed with the data written into it. Therefore to set a bit, write a "1" logic value to the desired bit location. To leave the bit value intact, write a logic "0".
- DCNTCL Clear bits – write. When writing to this address, the register's old value is AND'ed with the data written into it. Therefore, to clear a specific bit, write a logic "0" to the desired bit location. To retain the bit's original value, write a logic "1".

See "Direct Memory Access" on page 7-1 for complete DMA register definitions.

External Port DMA Register

Table 2-15. External Port DMA Register

Register Quad	External port DMA TCBs (DMA channels 0 - 3)	Direct Memory Address	Reset value	Remarks
DCS0	DMA channel 0 source TCB	0x180400 0x180401 0x180402 0x180403	0xD300 0000 0x0000 0000 0x0100 0004 0x0000 0000 or 0x0	1, 2
DCD0	DMA channel 0 destination TCB	0x180404 0x180405 0x180406 0x180407	0x5300 0000 0x0000 0000 0x0100 0004 0x0000 0000	1, 2
DCS1	DMA channel 1 source TCB	0x180408-B		2
DCD1	DMA channel 1 destination TCB	0x18040C-F		2
DCS2	DMA channel 2 source TCB	0x180410-3		2
DCD2	DMA channel 2 destination TCB	0x180414-7		2
DCS3	DMA channel 3 source TCB	0x180418-B		2
DCD3	DMA channel 3 destination TCB	0x18041C-F		2

- 1 According to Boot Mode strap.
- 2 DMA registers can be accessed only as quad-words.

AutoDMA Registers

The AutoDMA registers are used to implement a slave mode DMA (see “Direct Memory Access” on page 7-1). These registers can only be accessed through multiprocessing memory.

Register Access Features

Table 2-16. AutoDMA Register

Register Quad	AutoDMA Registers	Direct Memory Address	Remarks
AUTODMA0	AutoDMA register 0	0x180740 - 0x180743	
AUTODMA1	AutoDMA register 1	0x180744 - 0x180747	

AutoDMA0 Register

This is the AutoDMA channel 0 Data register. When writing to this register, DMA channel 12 transfers the written data into the internal memory address programmed in the DMA. This register cannot be read, and can only be written through the multiprocessing address space. If the DMA is not initialized, the data is lost.

AutoDMA1 Register

This is the AutoDMA channel 1 Data register. When writing to this register, DMA channel 13 transfers the written data into the internal memory address programmed in the DMA. This register cannot be read, and can only be written through the multiprocessing address space. If the DMA is not initialized, the data is lost.

Link Port Transmit DMA Register

Table 2-17. Link Port Transmit DMA Register

Register Quad	Link Output DMA TCBs (DMA channels 4, 5, 6, 7)	Direct Memory Address	Remarks
DC4	DMA channel 4 TCB	0x180420 - 3	1
DC5	DMA channel 5 TCB	0x180424 - 7	1
DC6	DMA channel 6 TCB	0x180428 - B	1
DC7	DMA channel 7 TCB	0x18042C - F	1
	Reserved	0x180430 - 3	

1 DMA registers can be accessed only as quad-words.

Link Port Receive DMA Register

Table 2-18. Link Port Receive DMA Register

Register Quad	Link Input and IFIFO DMA TCBs (DMA channels 8, 9, 10, 11, 12, 13)	Direct Memory Address	Reset Value ¹
DC8	DMA channel 8 TCB	0x180440	0x5780 0000
		0x180441	0x0000 0000
		0x180442	0x0100 0004
		0x180443	0x0000 0000
DC9	DMA channel 9 TCB	0x180444	0x5780 0000
		0x180445	0x0000 0000
		0x180446	0x0100 0004
		0x180447	0x0000 0000
DC10	DMA channel 10 TCB	0x180448	0x5780 0000
		0x180449	0x0000 0000
		0x18044A	0x0100 0004
		0x18044B	0x0000 0000

Register Access Features

Table 2-18. Link Port Receive DMA Register (Cont'd)

Register Quad	Link Input and IFIFO DMA TCBs (DMA channels 8, 9, 10, 11, 12, 13)	Direct Memory Address	Reset Value ¹
DC11	DMA channel 11 TCB	0x18044C	0x5780 0000
		0x18044D	0x0000 0000
		0x18044E	0x0100 0004
		0x18044F	0x0000 0000
	Reserved	0x180450 - 3	
DC12	DMA channel 12 TCB	0x180458	0x5780 0000
		0x180459	0x0000 0000
		0x18045A	0x0100 0004
		0x18045B	0x0000 0000
DC13	DMA channel 13 TCB	0x18045C	0x5780 0000
		0x18045D	0x0000 0000
		0x18045E	0x0100 0004
		0x18045F	0x0000 0000

1 DMA registers can be accessed only as quad-words.

DMA Control and Status Register

The Status register *DSTAT* is read only, and has two addresses. When reading the Status register from the clear bits address *DSTAT*, all the error codes in it are cleared, and changed to IDLE state *DSTATC*.

Table 2-19. DMA Control and Status Register

Register Quad	DMA Control)	Direct Memory Address	Reset value
DCNT	DMA control register	0x180460	0x0 ¹
DCNTST	DMA control register set bits	0x180464	1
DCNTCL	DMA control register clear bits	0x180468	1
DSTAT	DMA status register	0x18046C - F	Read only ²
DSTATC	DMA status register clear bits	0x180470 - 3	Read only ²

- 1 DMA control registers (DCNT, DCNTST, DCNTCL) can be accessed as normal, long, or quad-words.
- 2 DMA status registers (DSTAT, DSTATC) can be accessed as long or quad-words.

Link Registers

The Link Port Buffer registers are only accessible as quad-words. There are two Link register groups described in Table 2-17 on page 2-45 and Table 2-18 on page 2-45. Link Control and Status registers can be accessed only as single-words. See “Link Ports” on page 8-1 for complete Link register definitions.

Link Port Control and Status Register

Table 2-20. Link Port Control and Status Register

Register Quad	Reserved for Links 4-5 Registers and Control Registers	Direct Memory Address	Reset Value
LCTL0	Link # 0 control register	0x1804E0	0x400
LCTL1	Link # 1 control register	0x1804E1	0x400
LCTL2	Link # 2 control register	0x1804E2	0x400
LCTL3	Link # 3 control register	0x1804E3	0x400
LSTAT0	Link # 0 status register	0x1804F0	Read only
LSTAT1	Link # 1 status register	0x1804F1	Read only
LSTAT2	Link # 2 status register	0x1804F2	Read only
LSTAT3	Link # 3 status register	0x1804F3	Read only
LSTATC0	Link # 0 status clear register	0x1804F8	
LSTATC1	Link # 1 status clear register	0x1804F9	
LSTATC2	Link # 2 status clear register	0x1804FA	
LSTATC3	Link # 3 status clear register	0x1804FB	

Register Access Features

Link Port Receive and Transmit Buffers

Table 2-21. Link Port Receive and Transmit Buffers

Register Quads	Links 0 - 3 Registers	Direct Memory Address	Remarks
LBUFTX0	Link # 0 Tx register	0x1804A0-3	-
LBUFRX0	Link # 0 Rx register	0x1804A4-7	Read only
LBUFTX1	Link # 1 Tx register	0x1804A8-B	
LBUFRX1	Link # 1 Rx register	0x1804AC-F	Read only
LBUFTX2	Link # 2 Tx register	0x1804B0-3	-
LBUFRX2	Link # 2 Rx register	0x1804B4-7	Read only
LBUFTX3	Link # 3 Tx register	0x1804B8-B	-
LBUFRX3	Link # 3 Rx register	0x1804BC-F	Read only

3 CORE CONTROLS

This chapter discusses clocking inputs, TigerSHARC processor's three operating modes (and low power mode), and the boot modes that initiate the TigerSHARC processor. The chapter also includes a discussion of Flag and Timer pins.

The operating modes are:

- Emulation mode
- Supervisor mode
- User mode

The boot modes are:

- EPROM boot
- Link boot
- Boot by other master
- No boot

Clock Inputs

The TigerSHARC processor has two primary clock inputs—local clock (LCLK) and system clock (SCLK). These clocks must receive the same input (tie the SCLK_P and LCLK_P pins together). These clock signals can be the


Operation Modes

same output from the same clock buffer or the output from two different buffers driven by the same initial clock source. Both inputs are differential.

The SCLK is the cluster bus interface clock. The cluster bus signals are specified in the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet* with respect to this clock.

The LCLK is the input clock to an Analog Phase Locked Loop (PLL) that generates the internal clock CCLK. All the internal descriptions (pipeline, internal bus, and so on.) use CCLK as the reference clock. The frequency of CCLK is the frequency of LCLK multiplied by a constant value defined by the input pins LCLKRAT2–0. The constant can be 2, 2.5, 3, 3.5, 4, 5, or 6.

In a system that must work deterministically cycle-by-cycle, only an integer LCLK multiplication (2, 3, 4, 5, or 6) should be used when setting LCLKRAT.

 For more information on using clock ratios, see the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*.

Operation Modes

The TigerSHARC processor can operate in one of three modes: emulation, supervisor, or user. In each of these modes, *all* instructions are executed normally. The current operating mode of the TigerSHARC processor affects which components of the processor are active and can be accessed, as well as which exceptions are taken, and how they are handled. The modes are described from highest to lowest priority.

Emulation Mode

Emulation mode is used when controlling the processor with an emulator tool via the JTAG port. The TigerSHARC processor enters emulation mode when an emulation exception is generated. An emulation exception is generated after any one of these events:

- EMUTRAP instruction
- Watchpoint when programmed to cause emulation trap
- JTAG private instruction
- TMS change while TEME bit in EMUCTL is set

Emulation exceptions are the highest priority exceptions or interrupts.

While the TigerSHARC processor is in emulation mode, the only source of instructions is the EMUIR register. The EMUIR register is loaded via the JTAG Test Access Port (TAP). When entering this mode, the external JTAG controller (Analog EZ-ICE or other customer hardware) must be enabled. For more information, see “Sequencer Control Register – SQCTL” on page 2-16. Once the emulation features are enabled and an emulation exception is encountered, the TigerSHARC processor enters emulation mode. Once the TigerSHARC processor is operating in emulation mode, the only way it can exit emulation mode is by executing a return from interrupt (RTI).

In emulation mode, the debug registers (Ureg group 0x1B) can be accessed only by move register-to-register or immediate data load instructions. These registers cannot be loaded from or stored to memory directly.

Operation Modes

In emulation mode, the program access to the debug registers (groups 0x1B and 0x3D) may be executed only using the instruction:

```
UREG = UREG;
```

The BTB, cycle count, performance monitors, and trace buffer are inactive in emulation mode. Except for `if true—RTI(np)`, *all* control flow instructions (`jump`, `call`, `RDS`, `conditional`, and so on.) and the `IDLE` instruction cannot be used in emulation mode. The Run Test /Idle (RT/I) register can only be used without condition.

If the external reset pin is being asserted, and an external emulation exception is generated, the TigerSHARC processor core exits reset internally, enters emulation mode, and waits to fetch the value of `EMUIR`, which is loaded from the JTAG.

In emulation mode, only the core works. The external interfaces (BIU, DMA and links) are still held in reset, although their internal registers can be accessed by the instructions inserted via JTAG and `EMUIR`. This feature enables an EZ-ICE to initialize the TigerSHARC processor internal registers and memory while it is still in reset and start its run from a known state.

Supervisor Mode

Most code that is not intended to run under an operating system should be designed to run in supervisor mode. Supervisor mode allows the program to access all resources. The TigerSHARC processor is in supervisor mode when one of these two conditions is true:

- The `NMOD` bit in `SQCTL` is set. For more information, see “Sequencer Control Register – `SQCTL`” on page 2-16.
- An interrupt routine is executed—indicated by non-zero `PMASK`. For more information, see “`PMASK` Register” on page 4-13.)

Normally when the TigerSHARC processor is reset (via hardware or software), it is idle. It exits the idle state to a running state when an interrupt is issued. The interrupt puts the TigerSHARC processor into supervisor mode.

If the `NMOD` bit in `SQCTL` is cleared, the processor enters user mode after leaving an interrupt routine (unless it is nested inside another interrupt routine).

User Mode

The user mode operation is used for algorithms and control code that does not require the manipulation of system resources. Many system resources are not accessible in user mode. If the TigerSHARC processor attempts to access these resources, an exception occurs.

User mode is often used when running code out of an operating system. The operating system kernel runs in supervisor mode, but the user code is restricted to user mode.

The registers that may be accessed by core program in user mode are:

- Groups 0x00 to 0x09 – Compute block registers
- Groups 0x0C to 0x0F – IALU registers
- Sequencer registers – `CJMP`, Loop counters `LC0` and `LC1`, and the Static Flag register (`SFREG`)

All other registers cannot be written by the core program in user mode. An attempt to write to a protected register causes an exception. These registers can still be accessed by another master (DMA, external host, and so on).

Low Power Mode

The TigerSHARC processor can enter sleep mode, where the TigerSHARC processor does not operate and power consumption is minimal. This feature is useful for systems that require a low power standby mode. Low power mode is entered as described below. The TigerSHARC processor returns to normal operation after an external interrupt (falling edge on the $\overline{\text{TRQ}}$ pin) occurs.

Entering Low Power Mode

The TigerSHARC processor enters into low power mode using the `IDLE(LP); ;` instruction. The procedure to follow in order to enter low power mode depends on the processor system. This procedure is described in the following sections.

Single Processor System

In a single processor system, low power mode can be entered after the following procedure.


1. Ensure that the interrupt that wakes the TigerSHARC processor is enabled.
2. Stop all the DMA channels. This can be done by setting all the `PAUSE` bits in the `DCNT` register. For more information, see “DCNT Register” on page 7-26.
3. Check that the link is not in the middle of a transmit or receive transaction. If it is controlled by the DMA, it stops at the end of the quad-word; otherwise the application can control it.
4. Check that the external bus OFIFO and IFIFO are empty. DMA transactions are not be issued after step 2 is executed. To check that all the transactions are complete and to ensure that the OFIFO and IFIFO are empty, initiate a write to an internal address via the

multiprocessing space. After this instruction is issued, no external access should be driven by the program. When the data is written to internal space, the OFIFO is empty.

5. The `IDLE(LP)`: instruction can now be executed.

Multiprocessing Systems

In a multiprocessing system, the procedure for entering low power mode is more complex.

 All the TigerSHARC processors in the system should go into low power mode together. Do not attempt to put some of the TigerSHARC processors in the cluster into low power while others are still working. This may cause the arbitration system to go into deadlock.

One of the TigerSHARC processors, or the host (designated the Low Power Driver or LPD), should be selected as the *last* to go into low power and the *first* to wake. The procedure is as follows.

1. The LPD TigerSHARC processor should lock the bus by setting the `BUSLK` bit in its `BUSLK` register (see “`BUSLK` System Control” on page 2-38) and wait until it has control of the bus. If the host is the LPD, it must set the bus lock bit.
2. The LPD TigerSHARC processor follows *steps 1 to 3* in the flow for a single processor system. If the LPD is host, the steps follow the host configuration routine.
3. The LPD TigerSHARC processor indicates to the other TigerSHARC processors they can enter low power. The indication can be sent through an interrupt, flag pin, or a write transaction on the cluster bus. After this transaction, the LPD can no longer access the other processors in the system.

Flag Pins

4. After the other TigerSHARC processors receive the indication from the LPD, they should follow *steps 1 to 3*, and 5 in the flow for a single processor system.
5. The LPD now executes *steps 4 and 5* in the flow for a single processor system. This can also run in parallel to step 4 above.

Return to Normal Operation

The TigerSHARC processor returns to operation when it senses a falling edge on any of the interrupt inputs $\overline{\text{TRQ3-0}}$. The interrupt, if enabled, causes the TigerSHARC processor to execute the interrupt routine.

In a multiprocessing system, all TigerSHARC processors should be activated by the interrupt. The LPD wakes up as a master with the bus locked. It cannot release the bus lock or access the other TigerSHARC processors until all TigerSHARC processors are active. This can be indicated by the flag pins.

Flag Pins

There are four input/output flag pins. Each pin can be individually configured to be an input or output. When they are configured as input flag pins, they can be used either as a condition or as a value in the SQSTAT register. (See “Sequencer Status Register – SQSTAT” on page 2-16.) After power up reset, $\text{FLAG}(3-0)$ are inputs where static 100K Ω pull-downs hold them at zero value.

Timers

The TigerSHARC processor has two general-purpose 64-bit timers—Timer 0 and Timer 1. The timers are free-running counters that are loaded with an initial value and give an indication when expiring. The indication is normally an interrupt, but could also be an external pin for Timer 0.

Timer Registers

Each timer is composed of two long registers. One register is the initial value (`TMRINxx`) and the other is the running value (`TIMERxx`). The `TMRINxx` register is read/write and the `TIMERxx` is read-only. When the value of `TIMERxx` is read, the point in time at which it is read is uncertain. Consequently, results may vary.

Like all the sequencer registers, the timer registers can be accessed only by a single-word access.

Two bits in the sequencer control register enable the timer count—`TMRORN` for Timer 0 and `TMR1RN` for Timer 1.

Timer Operations

The timers are initialized to idle status after reset (achieved by clearing the `TMRiRN` bits in `SQCTL`). To set the timers to run, the application has to set the `TMRiRN` bit. When this bit is set, the value in `TMRINxx` is copied to the timer registers, and these start counting down, one count every internal clock cycle.

Whenever the timer count reaches zero, the timer issues the two timer expire interrupts (high and low priority), reinitializes the counter to the initial value, and starts running again. If the timer is active (`TMRiRN` bit is

Timers

set), writing 1 to `TMRINxx` has no effect. Writing a different value to `TMRINxx` while the timer is operating changes the setup after the next expire of the timer (when `TMRINxx` value is copied again to the timer).

Timer 0 Output Pin

Pin `TMROE` indicates Timer 0 expire and issues a pulse on the timer out pin for four `SCLK` cycles.

4 INTERRUPTS

The TigerSHARC processor supports several types of interrupts— internal and external interrupts can serve any of the following purposes.


- For synchronization between core and non-core operations
- For error detection
- For using debug features
- For introducing control by an application

Most interrupts in the TigerSHARC processor are dedicated. Four interrupt pins and one interrupt register support general-purpose interrupts. All interrupt sources other than the interrupt pins and interrupt register are caused by events or dedicated hardware. For each interrupt there is a vector register in the Interrupt Vector Table (IVT) (in register groups 0x38 and 0x39) and a bit number in the interrupt flags and mask registers.

For performance considerations, when a hardware interrupt occurs, it does not break the pipeline. The first instruction of the interrupt routine is inserted into the instruction flow after it has occurred. From this point, the instructions that are already in the pipeline complete their execution.

Interrupt I/O Pins

Since it is possible to enter the interrupt service routine (ISR) immediately after clearing the IMASK register, the IMASK bit would be 0 while in the ISR. This does not affect operation, so it could be used to avoid working the interrupt.

 Software interrupts that are caused by a specific instruction must occur immediately after the instruction that caused them. Therefore the pipeline is flushed after a software interrupt, and the instructions that were in the pipeline are not executed.

Interrupt I/O Pins

To issue an interrupt, the Interrupt Request pins, $\overline{TRQ3-0}$, are asserted. Each of the $\overline{TRQ3-0}$ pins may be individually set to either edge triggered or level-sensitive. See “Sequencer Control Register – SQCTL” on page 2-16.

Interrupt Vector Table

Each of the interrupts implemented in the TigerSHARC processor has an interrupt vector register, which is the address of the interrupt routine that serves the appropriate interrupt. The whole register file (31 entries with another 32 reserved entries) is referred to as the Interrupt Vector Table (IVT). The registers are 32 bits each, in order to enable internal and external memory interrupt routines.

For boot procedures, some of the interrupt vector registers are initialized at reset to specific addresses. In boot by EPROM and boot by link operations, a DMA is initialized to load the boot data into memory address 0x0. When the boot DMA is executed, it issues a DMA interrupt. The DMA interrupt vector is also initialized to address 0x0.

Interrupt Types

Interrupts are classified as either *software interrupts* or *hardware interrupts*.

Software interrupts are caused by a specific instruction. Other instructions left in the pipeline are not be completed after the instruction that has caused the software interrupt. In most cases, performance optimization is not important for software interrupts.

Hardware interrupts are more performance critical because in many cases the application is based on the interrupts. Normally, the hardware interrupts have higher priority. When a software interrupt (exception or emulation interrupt) occurs, all instructions in the pipeline *after* the instruction that caused the exception are executed and all the sequential instructions are aborted. The following subsections describe the interrupt types, noting the corresponding interrupt bit in the `ILAT` and `IMASK` interrupt registers.

Level or Edge Interrupts

Every interrupt is defined as either level-sensitive or edge-sensitive—the exception being the \overline{TRQ} pins that can be either level- or edge- sensitive according to the programming in the `SQCTL` register. In “Interrupt Types” on page 4-3, the type (edge or level) is listed for each interrupt type. The differences between the two types of interrupts are:

- Edge-sensitive interrupts are latched when they occur and remain latched until serviced or reset by an instruction. If it is not cleared during servicing, no new event is identified and the continued request is ignored.
- Level-sensitive interrupt requests must be sustained until serviced, otherwise they are not seen. If the request continues after being serviced, it is considered a new interrupt.

Interrupts Generated by On-Chip Modules

A level-sensitive interrupt is normally driven as a result of some accessible register state and cleared either by reading the register or by writing an inactive value into it.

An edge-sensitive interrupt is event triggered (for example, by a timer).

Interrupts Generated by On-Chip Modules

Interrupts generated by on-chip modules differ depending on the specific TigerSHARC processor configuration.

Timers

The vector registers are:

- `IVTIMER0HP` – interrupt priority 52; edge-triggered
- `IVTIMER1HP` – interrupt priority 53; edge-triggered
- `IVTIMER0LP` – interrupt priority 2; edge-triggered
- `IVTIMER1LP` – interrupt priority 3; edge-triggered

After reset, the timer interrupts are disabled and vectors are not initialized.

There are four timer interrupts: two for each timer, one as high priority and one as low priority. When the timer interrupt occurs, both high and low priority bits are set in `ILAT` (for example, for Timer 0 both Bit2 and Bit52). The purpose of having the two priorities per timer is to enable the user to choose which priority is needed. This is accomplished by enabling the appropriate interrupt. Only one bit is cleared when the interrupt routine is served. If both high and low priority interrupts are enabled, the interrupt is serviced twice. The assumption is that only one of the interrupts is enabled.

Link Interrupts

The vector registers are:

- IVLINK0 – interrupt priority 6; level-triggered
- IVLINK1 – interrupt priority 7; level-triggered
- IVLINK2 – interrupt priority 8; level-triggered
- IVLINK3 – interrupt priority 9; level-triggered

After reset, the link interrupts are disabled and vectors are not initialized.

There are four link channels that normally work with their dedicated DMA channels. While there is a data element in the receive buffer and its DMA is not initialized, this link issues an interrupt.

DMA Interrupts

The vector registers are:

- IVDMA0 – interrupt priority 14; edge-triggered
- IVDMA1 – interrupt priority 15; edge-triggered
- IVDMA2 – interrupt priority 16; edge-triggered
- IVDMA3 – interrupt priority 17; edge-triggered
- IVDMA4 – interrupt priority 22; edge-triggered
- IVDMA5 – interrupt priority 23; edge-triggered
- IVDMA6 – interrupt priority 24; edge-triggered
- IVDMA7 – interrupt priority 25 edge-triggered
- IVDMA8 – interrupt priority 29; edge-triggered

Interrupts Generated by On-Chip Modules

- IVDMA9 – interrupt priority 30; edge-triggered
- IVDMA10 – interrupt priority 31; edge-triggered
- IVDMA11 – interrupt priority 32; edge-triggered
- IVDMA12 – interrupt priority 37; edge-triggered
- IVDMA13 – interrupt priority 38; edge-triggered

After reset, the DMA interrupts are enabled and vectors are initialized to zero for boot purposes.

There are 14 DMA channels, each of which can generate an interrupt. If the interrupt bit in its TCB is set, an interrupt is set when the appropriate channel completes its block.

Interrupt Pins (IRQ)

The vector registers are:

- IVIRQ0 – interrupt priority 41, edge-/level-triggered, programmable, init value: 0x10000000
- IVIRQ1 – interrupt priority 42, edge-/level-triggered, programmable, init value: 0x08000000
- IVIRQ2 – interrupt priority 43, edge-/level-triggered, programmable, init value: 0x0C000000
- IVIRQ3 – interrupt priority 44, edge-/level-triggered, programmable, init value: 0x0

The interrupt type is edge- or level-triggered according to SQCTL programming. See “Sequencer Control Register – SQCTL” on page 2-16.

After reset, the $\overline{\text{IRQ}}$ interrupts are disabled, unless $\overline{\text{IRQEN}}$ strap option (on pin $\overline{\text{BM}}$) is used. Vectors are initialized for boot purposes.

There are four general-purpose interrupt pins that can be programmed to be edge- or level-sensitive. When one of these interrupt pins is activated, an interrupt is issued (if enabled).

Vector Interrupt (VIRPT)

The vector register is:

VIRPT – interrupt priority 48; edge-triggered

After reset, the vector interrupt is enabled but the vector is not initialized. This is one of the events that can initiate booting by another master. Additionally, this interrupt is a general-purpose interrupt for another master's use. Another master, either the host or a TigerSHARC processor, can write an address to this register. The write causes a vector interrupt to occur. The value that is written into the register is the address of the interrupt routine.

Bus Lock Interrupt

The vector register is:

- IVBUSLK – interrupt priority 50; edge-triggered

After reset, the bus lock interrupt is disabled and vectors are not initialized.

This interrupt is issued when the bus lock bit in the SQCTL register is set and the TigerSHARC processor becomes the bus master. This interrupt is used to indicate that the TigerSHARC processor has locked the external bus.

Hardware Error Operations

Hardware errors generate actions defined by the vector register as follows.

- **IVHW** – interrupt priority 57, level-triggered, active as long as any of the status registers set by events in the list below indicate an error (by issuing an **HWILL** interrupt).
- **Error in DMA** – one of the **CHxx** fields in **DSTAT** register is 100 (TCB initialization error). Refer to “DMA Status Register (DSTAT/DSTATC)” on page 7-23.
- **Data written to the AutoDMA while the corresponding AutoDMA channel is not initialized.** Active while Bit17 of **SYSTAT** is active. Refer to the register description for “SYSTAT/SYSTATCL Register” on page 2-31.
- **Broadcast read from external.** Active while Bit16 of **SYSTAT** is active. Refer to the register description for “SYSTAT/SYSTATCL Register” on page 2-31.
- **Access to SDRAM when it is not enabled.** Active while Bit18 of **SYSTAT** is active. Refer to the register description for “SYSTAT/SYSTATCL Register” on page 2-31.
- **Self multiprocessing read** – active when Bit19 of **SYSTAT** is active. Refer to the register description for “SYSTAT/SYSTATCL Register” on page 2-31.
- **Link error** – in one of the **LSTATx** registers in the fields **RER** or **TER** indicate an error. Refer to “Error Detection Mechanisms” on page 8-17.

For additional information refer to “Status Register (LSTATx)” on page 8-23).

After reset, the hardware error interrupt is disabled and vectors are not initialized.

Software Exceptions

Software exceptions are interrupts stemming from code execution.

The vector register is:

- IVSW – interrupt priority 62



If an exception occurs when the interrupt is disabled, the exception is lost.

After reset, the software exception is disabled and vectors are not initialized.

Software exceptions include:

- Floating-point exceptions defined by the IEEE standard. When these occur, a software interrupt is executed.
- Underflow if UEN in the XSTAT/YSTAT register is set
- Overflow if OEN in the XSTAT/YSTAT register is set
- Debug exceptions that cause supervisor exceptions:
 - TRAP instruction
 - Watchpoint match, when programmed to cause an exception
- Illegal operations:
 - Undefined instruction (not on all cases)
 - Illegal combination of instructions, described in the “Instruction Flow” chapter of the *ADSP-TS101 TigerSHARC Processor Programming Reference*
- External access to an illegal space – multiprocessing broadcast read access by either load/store instruction or by fetch

Software Exceptions

- Illegal misalignment of load – not DAB accessible due to odd address for long-word; or address that is not quad-aligned for quad-word
- Protection violation – access to a supervisor register in user mode
- Two instructions in the same line with one destination
- More than three results in a single compute instruction line (for example, add and subtract, multiply, shift)
- Two 32-bit immediate instructions in the same line
- More than one branch in the same aligned quad-word with prediction bit set

Emulation Debug

The vector register is:

- `IVDBG`, no register – after emulation debug exception the TigerSHARC processor fetches from `EMUIR`.

The emulation debug exception causes the TigerSHARC processor to go into emulation mode. In this mode, instructions are read from the JTAG interface and executed in a single step. Emulation mode can be caused by:

- An `EMUTRAP` instruction
- A watchpoint match (when programmed to cause an emulation trap)
- A JTAG emulation activation

During and after reset, the emulation trap is enabled.

Other Interrupt Registers

The interrupt controller includes three interrupt registers:

- ILAT registers – ILATH and ILATL
- IMASK registers – IMASKH and IMASKL
- PMASK registers – PMASKH and PMASKL

The IMASK, ILAT, and PMASK registers all have the same bit definitions as described in the “Programmer Sequencer” chapter of the *ADSP-TS101 TigerSHARC Processor Programming Reference*.

ILAT Register

The ILAT register is a single 64-bit register accessed as two 32-bit registers, ILATL and ILATH. Each bit is dedicated to an interrupt—when an interrupt occurs, the corresponding bit is set. The order of interrupt bits is the interrupt priority—Bit0 is the lowest priority. See “IMASK Register” on page 2-19 for bit assignments.

An application may emulate an interrupt by writing to the ILAT register set address (ILATSTL or ILATSTH). Writing to these addresses updates the ILAT register to the OR of the old value and the written value. As a result, every set bit in the written data sets the corresponding bit in the ILAT register. Setting an interrupt bit causes the TigerSHARC processor to assume the corresponding interrupt has occurred.

Interrupt bits can also be cleared by writing to the clear addresses (ILATCLL or ILATCLH). Such a write ANDs the data written with the old data of the ILAT register. In this case, a zero bit in the input data clears the corresponding bit in ILAT, while a set bit keeps it unchanged. This way, an

Other Interrupt Registers

application can clear a pending interrupt before it is executed. Such an operation is very sensitive, and it should be executed under these conditions:

- Non-implemented bits may not be set. When writing to either set or clear address, the reserved bits must be zero.
- Exception and emulation exception (Bits 62 and 63) may not be set. In order to cause an exception, use a `TRAP` instruction; in order to cause an emulation exception, use an `EMUTRAP` instruction.
- Interrupts that are level-triggered should not be changed—link interrupts, hardware error interrupts, or \overline{TRQ} bits if programmed to be level-triggered.
- Writes to `ILATL` or `ILATH` should not be attempted.
- An interrupt should be cleared while it is masked, otherwise it may be served before it is cleared.
- Like all other sequencer registers, the set and clear addresses are single-word writes only.

IMASK Register


The `IMASK` register is a single 64-bit register accessed as two 32-bit registers, `IMASKL` and `IMASKH`. Mask bits (Bit63–61 and Bit59–0) are individually dedicated to different interrupts. When an interrupt bit is set in the `ILAT` register, the corresponding interrupt is only accepted if the corresponding bit is also set in the `IMASK` register. Consequently, the order of the interrupt bits is identical to those found in the `ILAT` register. Bit60 is a global hardware interrupt enable. When cleared, no interrupts, except for exception and emulation, are enabled.

PMASK Register

The `PMASK` register is a single 64-bit register that is accessed as two 32-bit registers, `PMASKL` and `PMASKH`. This register is identical to the `IMASK` register in bit assignment.

These registers help the hardware track nested interrupts by masking all interrupts with a lower or equal priority than the interrupt that is currently being executed. The `PMASK` is a 64-bit register, of which 32 bits are implemented and 32 are reserved. When the TigerSHARC processor begins to service an interrupt, its bit in `PMASK` is set. Each set bit indicates this ISR is active or nested at some level. The most significant set bit in `PMASK` indicates which interrupt is currently served. The current interrupt bit is cleared once the service is completed, either by `RTI` instruction or by `RDS` instruction (see “if cond, `RTI (ABS)`” and “if cond, `RDS`” in the *ADSP-TS101 TigerSHARC Processor Programming Reference*).

When the `PMASK` is nonzero, all the interrupts with a lower or equal priority to the most significant set bit in `PMASK` are disabled by it. For logic equations, the `PMASK_R` is defined as the mask created by `PMASK`; that is, all the bits above the most significant set bit are set and other bits are clear. A `PMASK_R` is only a definition and not a register.

 The `RETI` register is updated only in the E2 pipe stage of the first instruction of the interrupt service routing. The first instruction cannot be an instruction that uses its value, such as `RTI` or `RETI`.

Interrupt Service

Example

During normal operations the `PMASK` is zero, and `PMASK_R` is `0xFF...F`.

1. Assume that a link #0 interrupt occurred.
`PMASK = 0x00...040`, and `PMASK_R = 0xFF...F80`. The interrupts of lower priority than link #0—namely, timer low priority interrupts and link #0 interrupt—are disabled by `PMASK`.
2. Following this, assume a DMA #2 interrupt occurred.
`PMASK = 0x00...08040`, and `PMASK_R` is `0xFF...F0000`. The timer low priority, link, and DMA #0, #1 and #2 interrupts are disabled by `PMASK`. Returning from interrupt (which is DMA #2 interrupt) clears the most significant set bit in `PMASK` which is bit #15. The `PMASK` register is now `0x00...040`, and `PMASK_R` is `0xFF...F80`. The timer low priority interrupts and link #0 interrupt are disabled by `PMASK`, but the other link interrupts and DMA interrupts are enabled again.

Interrupt Service

When an interrupt occurs, the corresponding bit in the `ILAT` register is set. The `ILAT` bits are AND'ed with the enable bits in the `IMASK` and with `PMASK_R`. If the global interrupt enable bit is set and the result of the `ILAT` and `IMASK` and `PMASK_R` is not zero, the highest priority interrupt that is enabled is served.

The following algorithm describes the interrupt service routine.



In ISR the first instruction must not be `RDS` or `RTI`.

1. The interrupt vector in Interrupt Vector Table (IVT) is used as the next fetch address and the ISR's first instruction is pushed into the pipeline. If the interrupt is a hardware interrupt, normal execution continues until the first interrupt routine instruction reaches pipeline stage EX2. Otherwise, for software exceptions that are not hardware interrupts, all the normal flow instructions that are in the pipeline are aborted. While an interrupt is in the pipeline, all the hardware interrupts are disabled, although this is not reflected in the mask registers (`PMASK` or `IMASK`).

If the exception is disabled, the `ILAT` bit for the exception is not set and the exception is ignored. The reason is that the instruction line that caused the exception is lost.

2. Instructions in the pipe are executed.
3. Before the first instruction in the interrupt routine reaches EX2, the global enable interrupts (`IMASK60`, `PMASK60`) are rechecked. If the bit was cleared since the interrupt was generated by an instruction already in the pipeline, then all the instructions in the pipeline are aborted and the TigerSHARC processor begins fetching from the normal flow (as if there were no interrupt).



If instructions are ordered in a specific way, it is possible to enter an ISR immediately after it's bit is cleared in the `IMASK` register. While in the ISR, `IMASK` bit = 0, which does not affect operation in any way.

4. The return PC (which points to the instruction that would have been executed had the interrupt not occurred) is saved in the appropriate register—`RETI` for a hardware interrupt, `RETS` for a software interrupt, and `DBG_E` for an emulation debug.

Interrupt Service

5. The TigerSHARC processor mode is changed to emulation mode if there was an emulation exception; otherwise it changes to supervisor mode.
6. When the first instruction in the interrupt routine reaches EX2 and if the interrupt is edge-triggered, the corresponding interrupt bit in the `ILAT` register is reset and the same bit in the `PMASK` register is set. The interrupt bit in `PMASK` is set in any case, not just when the interrupt is level-sensitive. Bit60 of `PMASK` is set only for hardware interrupts, but not in the case of an exception or emulation exception. When the interrupt bit in `PMASK` is set, it also disables all hardware interrupts until the `RETI` register is saved.

A software interrupt occurs only when the instruction that caused it reaches pipeline stage EX2. When a software interrupt that was enabled occurs, all the instructions in the pipeline after the instruction that caused the exception are aborted, including hardware interrupt routine instructions. If a software interrupt occurs during the time that the hardware interrupt is in the pipeline, the hardware interrupt is aborted with other instructions in the pipeline and the software interrupt procedure begins. The hardware interrupt flag is not reset, however, in order to save the (hardware) interrupt.

The hardware ISR steps, illustrated and numbered in Figure 4-1 on page 4-17, are as follows.

1. Determine state after hardware interrupt (`IMASK60 AND IMASKN AND PMASK_RN AND NOT IMASK60`).
2. Once masks are set:
 - `PMASK60` is embedded in `PMASK_R`, that is, if `PMASK60` is set then `PMASK_R50-0 = 0...0`
 - Exceptions and emulation conditions are not affected by `PMASK60` or `IMASK60`
 - Insert ISR fetch address

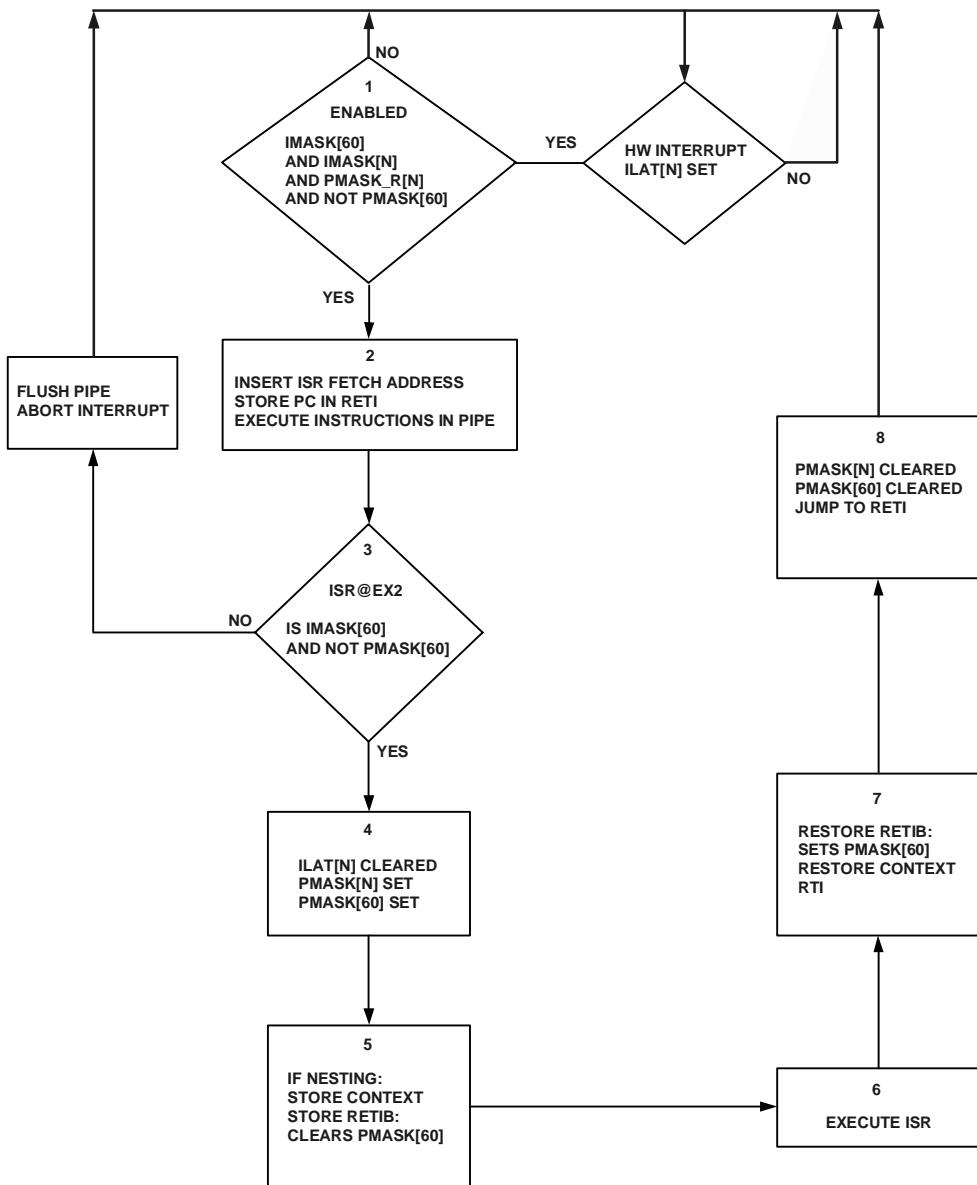



Figure 4-1. Hardware Interrupts

Interrupt Service

- Capture PC in order to store it in `RET1` when the instruction reaches `EX2`.

 Storing PC in `RET1` happens only after the first instruction of the interrupt routine reaches `EX2` with `ILAT` clear and `PMASK` bits set, if step 4 (below) conditions are met.

- Execute instructions in pipeline, unless it encounters an exception or emulation condition.

3. `ISR @ EX2`: Is `IMASK60 AND NOT PMASK60` set?


- If `NO` then flush pipeline, abort interrupt, and wait for next instruction.

4. If `YES`:

- `ILATN` cleared if interrupt is edge triggered
- `PMASK60` is set (`PMASK60` is embedded in `PMASK_R`, that is, `PMASK_R59-0 = 0...0`).
- Exceptions and emulation conditions are not affected by `PMASK60` or `IMASK60 PMASKN` set

5. If nesting, for hardware interrupt and `PMASK60` set, the `ISR` should:

- Store context
- Store `RETIB`: which automatically clears `PMASK60`

 Instructions for storing context and `RETIB` have to be specified in the software.

6. Execute `ISR`.

7. If the nesting was enabled, the ISR should:

- Restore RETIB: which automatically sets PMASK60
- Restore context
- RTI



Instructions for restoring RETIB and context have to be specified in the software.

8. Finally:

- PMASKN cleared
- If hardware interrupt, PMASK60 cleared
- Jump to RETI

Interrupt Handling

There are two methods of interrupt handling: one that uses nesting and one that does not. If the interrupt handling does not use nesting, other interrupts are disabled throughout the ISR or handling routine, because Bit60 in PMASK is set. If the interrupt handling routine does enable nesting, some special programming is required. The special programming required for nested interrupt handling routines is described in this section.

The first instructions in the interrupt routine must be dedicated to preserving the current status of the machine. All the registers used by the interrupt routine must be pushed onto a stack. The TigerSHARC processor does not require a dedicated stack pointer since any one of the IALU registers can be used as a stack pointer. The application can store and load registers with the post-modify option, and it can work with two stacks in two memories and store eight words every cycle.

RETIB is an aliased name for RETI. If you use RETIB, you access RETI while simultaneously setting or clearing PMASK60 (depending on whether you are writing or reading).

Interrupt Service

While preserving the current machine status in a stack, the appropriate global interrupt enable bit (hardware or software) is still clear, so there is no danger of corrupting the return address PC (RET1). If, however, the interrupt was a hardware interrupt, software or emulation exceptions can still be processed, since the return address is saved in a different register (RETS or DBGE). This also applies to emulation exceptions when a software interrupt is executing its initial cycles.

Once the relevant registers are saved in memory, the return register should be saved and the global interrupt disable bit `PMASK60` should be set. This operation is performed automatically by saving the return address alias `RETIB`.

If the system does not need to support nested hardware interrupts, there is no need to preserve the machine state in a stack. The return can be executed from the `RET1` register, and the hardware interrupt's global disable bit should be left as is. However, software interrupts or debug interrupts may be nested. In these cases, the machine state should be retained before software/debug service is performed.

The example below initializes the two timers and sets up their interrupts. The lower priority interrupt allows nesting, the higher one does not need to allow nesting.

```
.section data;
    .var register_store; /*internal memory location used to
store registers used in ISR*/
    .var register_store_2;

.section program; /*This example shows how to set up nested
interrupts using the two timers.*/

Setup_Timer_Interrupts:
/*Set up the interrupt service routines in the Interrupt Vector
Table*/
```

```

        j0 = j31 + TIMER0HP_ISR;; /* set TIMER 0 High Priority
interrupt vector*/
        IVTIMER0HP = j0;;
        j0 = j31 + TIMER1HP_ISR;; /* set TIMER1 High Priority
interrupt vector*/
        IVTIMER1HP = j0;;

/*Unmask Timer Interrupts*/
        xR0 = IMASKH;; /*Make sure that we are not clearing
other IMASK bits previously set*/
        xR1 = 0x10300000;;
        xR0 = R0 or R1;;
        IMASKH = xR0;; /* enable Global HW, TIMER0HP, and
TIMER1HP interrupts */

/* Trigger TIMER0HP Interrupt & TIMER1HP Interrupt*/
        TMRIN0H = 0x0;; TMRIN0L = 0xF;; TMRIN1H = 0x0;; TMRIN1L =
0xA;; /*Set timer lengths*/
        xR0 = 0x00003000;; /*Turn on Timer 0 and 1*/
        SQCTLST = xR0;;
Done: idle; nop; nop; nop;;
align_code 4;
jump Done (NP); nop; nop; nop;;

TIMER0HP_ISR: /*Lower priority of the two timers*/
        nop;; nop;; nop;; nop;;
        j0= j31+register_store;; /*Store registers that we are
going to use in the ISR*/
        q[j0+=4]=xr3:0;;
        xR0 = RETIB;; /*Clears Pmask[60], can now jump to higher
priority ISRs*/
        xrl=r3+r4;; /*Any code for the ISR would go here*/
        RETIB = xR0;; /*Sets Pmask[60], can no longer leave ISR for
other interrupts*/

```

Returning From Interrupt

```
RETIB = xR0;; /*Must do this twice because of anomaly*/
j0= J31+register_store;;
xr3:0=q[j0+=4];; /*restore registers*/
RTI (ABS)(NP); nop; nop; nop;;

TIMER1HP_ISR: /*Doesn't enable nesting because it is the higher
priority of the two interrupts*/
    nop;; nop;; nop;;
    j0= j31+register_store_2;; /*Store registers that we are
going to use in the ISR*/
    q[j0+=4]=xr3:0;;
    xr1=r3+r4;; /*Any additional operational code for the
ISR would go here*/
    j0= j31+register_store_2;;
    xr3:0=q[j0+=4];; /*restore registers*/
    RTI (ABS)(NP);;
```

Returning From Interrupt

The return from interrupt is performed using the `RTI` instruction (see “if cond, `RTI (ABS)`” in the *ADSP-TS101 TigerSHARC Processor Programming Reference*). The return address should be put in the `RETIB` register (recommended at about eight cycles beforehand to enable BTB usage). In non-nested hardware interrupt routines, the return address is already in the `RETI` register. If the non-nested exception is taken, the return address should be copied from the `RETS` or `DBGE` register (according to exception type) to the `RETI` register before the return. The use of the `RETIB` alias disables all interrupts by setting `PMASK60` until the `RTI` instruction is executed—this is to protect the `RETI` register from being destroyed by another interrupt occurrence.

If the system supports nested interrupts, the state of the machine is saved in memory at the beginning of the interrupt routine. The full state should be restored in the original registers before the return. See “Interrupt Handling” on page 4-19.

The `RTI` instruction clears the highest set bit in `PMASK` register. It also clears the global interrupt enable (`PMASK60`) if it is not an `RTI` from an exception or emulation exception. The instruction then jumps to the address pointed by the value in the `RETI` register and re-enables the interrupts.

Interrupt routine restrictions:

- An interrupt routine may not refer to the `RETI` or `RETS` registers on the first instruction line.
- An interrupt routine may not use the `RTI`, `RETI`, or `RDS` instructions on the first instruction line.

Exceptions

Exceptions are software interrupts—interrupts that are caused by code being executed. Figure 4-2 illustrates what transpires when a software interrupt is introduced. If the exception is enabled:

- `PMASK62` is set in case of a regular exception, or `PMASK63` is set in case of an emulation exception.
- The sequencer starts fetching from the address pointed to by the `IVSW` register in case of an exception, or from the `EMUIR` register in case of an emulation exception.
- `PC` is stored in `RETS` in case of an exception, or in `DBGE` register in case of an emulation exception.
- Instructions are flushed from the pipeline.

Exceptions

The return is executed by transferring the return address from RETS or DBGE (or from where it has been stored) to RETI without using RETIB. Then the RTI instruction should be executed.

If an exception occurs when it is not enabled, the exception is cleared and not serviced.

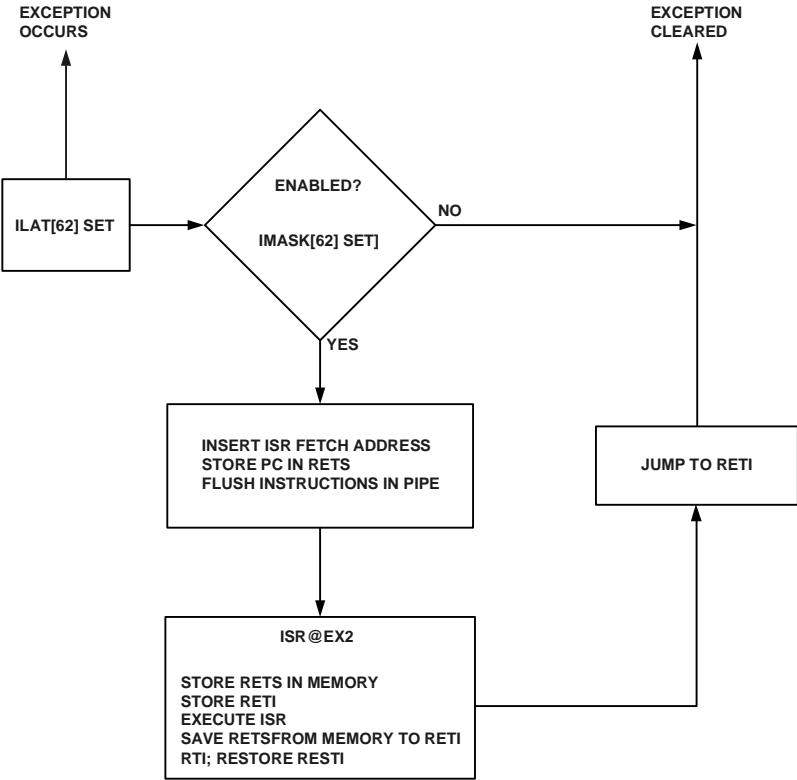


Figure 4-2. Software Exception (Interrupt)

The exceptions are treated differently from other interrupts:

- The exception is tied to the instruction that caused it. The exception process begins when the instruction that caused the exception reaches pipeline stage EX2. When this occurs, all instructions in the pipeline are aborted. If the instruction that caused the exception was speculative and it was aborted, the exception does not take place. Unlike an abort, a predicated instruction that is *not* executed due to the condition can still cause the exception.
- The return address is saved in a different register to enable nesting between the hardware interrupt and the exception. The registers are `RETS` for exception or `DBGE` for emulation trap.
- If the exception is disabled when it is occurring, it is not latched. It is not latched because it is irrelevant to do so without the return address pointing to the cause (`RETS` or `DBGS` registers). In these cases the exception is lost.
- When an exception occurs, Bit60 of `PMASK` is *not* set. `RTI` from an exception routine does not clear the `PMASK60`.

There are two reasons for this. First, an exception may occur while `PMASK60` is set and the value of `PMASK60` may not change due to the exception handling. Secondly, the purpose of protecting the `RTI` register from another interrupt is irrelevant because all hardware interrupts have a lower priority than (software) exceptions. Thus, there is no need to protect the `RTI` register before returning (by using `RETIB`).

- In order to protect the old `RETI` value before returning you must execute the following instruction lines, then jump to `RETS`:

```
[j31+temporary address] = RETI;;  
RETI = RETS;;  
RTI; RETI = [j31+temporary address];;
```

Exceptions

5 CLUSTER BUS

The TigerSHARC processor can be used as a single processor or as one element in a multiprocessor system. The processor can act as a standalone TigerSHARC processor or system, or it can be controlled by a host computer (for example, the TigerSHARC processor system can be an add-in board on a PC). The system architecture is flexible, and can be implemented according to the application requirements. The software “hooks” for the different options lie within the TigerSHARC processor. This chapter focuses on the external bus interface of the TigerSHARC processor, which includes the bus arbitration logic and the external address, data, and control buses.

The fastest protocol is the pipelined protocol. The TigerSHARC processor uses this protocol to interface with other TigerSHARC processors. The TigerSHARC processor can also use this protocol to interface with host and intelligent memory systems. The protocol has a peak performance of one data transfer every clock cycle and can sustain a throughput close to this peak performance because there is no restriction of address range for continuous flow in full throughput.

Another fast protocol is the SDRAM protocol. This protocol is defined by industry-standard SDRAM chips. The TigerSHARC processor has an on-chip SDRAM controller that drives all the SDRAM control signals (\overline{RAS} , \overline{CAS} , DWE , CKE , and DQM) and handles the initialization and refresh of the SDRAM chips. The SDRAM is useful for mass storage of the system, since it is possible to build very large memory arrays. The peak data throughput to the SDRAM is one data transfer every clock cycle. The

External Bus Features

throughput can be kept close to the maximum if sequential accesses are kept in the same page. This restriction is also applicable to block transfers by DMA. The overhead of a single access to SDRAM is high.

The TigerSHARC processor also supports slow device protocol. This protocol should be used for non-performance critical devices. In most systems, these devices should be connected on a secondary bus, since it may increase the load on the bus and slow the more critical accesses. It is possible, however, to put the slower devices directly on the external bus.

The external bus accommodates both 32-bit and 64-bit data bus widths and 32-bit address and control signals. Most signals are bidirectional since the TigerSHARC processor can be either the master or the slave on the external bus.

The external bus interface unit can be configured to many different setups by writing to the different external port control registers—`SYSCON` and `SDRCON` (see “Bus Control/Status (BIU) Register Group” on page 2-30). These two registers should be initialized shortly after reset, and cannot be updated during system operation. The setup of these two registers should be identical in all TigerSHARC processors in a given system. A broadcast write is recommended when initializing the `SYSCON` and `SDRCON` registers.

External Bus Features

The external bus includes:

- Bus width: 64- or 32-bits, configured separately for memory, multiprocessing or host interface
- Pipelined transactions with a programmable number of pipeline stages
- Programmable IDLE states

- Protocol supporting wait cycles inserted by the target slave, using the ACK pin
- EPROM and FLASH interface: 8-bit data bus with a fixed number of wait cycles, read or write
- Host interface
- SDRAM interface: no wait cycles required
- Support for slow I/O devices
- Glueless multiprocessing with other TigerSHARC processors, based on distributed bus arbitration
- Support of DMA transactions for external I/O devices through handshake mode
- Support for flyby between external memory and I/O in DMA

Bus Interface I/O Pins

Table 5-1 on page 5-4 lists the bus interface I/O pins. See Table 6-1 on page 6-7 for those I/O pins associated with SDRAM and Table 5-3 on page 5-38 for those associated with multiprocessing. Figure 6-3 on page 6-12 and Figure 6-2 on page 6-11 show the External Port Data Alignment.

Processor Microarchitecture

The TigerSHARC processor functions differently depending on the type of system in which it is used. It can perform as a single processor or in a multiprocessing system on a common external bus (as shown in Figure 5-1 on page 5-6). There may be a host (or host interface) in the system as well,

Processor Microarchitecture

Table 5-1. Bus Interface I/O Pins

Signal	Description
ADDR31-0	<p>Address Bus</p> <p>The TigerSHARC processor issues addresses for accessing memory and peripherals on these pins. In a multiprocessor system, the bus master outputs addresses for accessing internal memory or IOP registers of other TigerSHARC processors. The TigerSHARC processor inputs addresses when a host, or any other TigerSHARC processor, accesses its internal memory or Uregs.</p>
DATA63-0	<p>External Data Bus</p> <p>The TigerSHARC processor drives and receives data and instructions on these pins.</p>
\overline{RD}	<p>Memory Read</p> <p>\overline{RD} is asserted whenever the TigerSHARC processor reads from any slave in the system—excluding SDRAM accesses. When slave, \overline{RD} is an input indicating a READ on transactions that access the internal memory or Uregs. In a multiprocessing system, \overline{RD} is driven by the bus master. \overline{RD} changes concurrently with address pins during pipelined read accesses.</p>
\overline{WRL}	<p>Write Low</p> <p>\overline{WRL} is asserted in two cases: when the TigerSHARC processor writes to an even address word of the external memory or internal memory of another TigerSHARC processor, and when the TigerSHARC processor writes to a 32-bit zone (host, memory or TigerSHARC processor programmed to 32-bit bus).</p> <p>External master (host or TigerSHARC processor) asserts \overline{WRL} for writing to TigerSHARC processor's low word of internal memory. In a multiprocessing system, \overline{WRL} is driven by the bus master. \overline{WRL} changes concurrently with address pins during pipelined write accesses.</p>
\overline{WRH}	<p>Write High</p> <p>\overline{WRH} is asserted when the TigerSHARC processor writes a long-word (64 bits) or writes to an odd address word of the external memory or internal memory of other TigerSHARC processors on a 64-bit data bus. External master (host or another TigerSHARC processor) must assert \overline{WRH} for writing to the TigerSHARC processor's high word of 64-bit data bus. In a multiprocessing system, \overline{WRH} is driven by the bus master. \overline{WRH} changes concurrently with address pins during pipelined write accesses.</p>
ACK	<p>Acknowledge</p> <p>External devices can deassert ACK for adding wait states to external memory accesses. ACK is used by I/O devices, memory controllers and other peripherals on the data phase. The TigerSHARC processor can deassert ACK for adding wait states for synchronous accesses to its internal memory.</p>

Table 5-1. Bus Interface I/O Pins (Cont'd)

Signal	Description
$\overline{\text{BMS}}$	<p>Boot Memory Select</p> <p>$\overline{\text{BMS}}$ serves as chip select for boot EPROM or flash memory. If the TigerSHARC processor is configured to boot from EPROM, $\overline{\text{BMS}}$ is active during the boot sequence. In a multiprocessor system, $\overline{\text{BMS}}$ is driven by the bus master. During reset, the $\overline{\text{BMS}}$ is used as a strap pin for EPROM boot mode.</p>
$\overline{\text{MS1-0}}$	<p>Memory Select</p> <p>$\overline{\text{MS0}}$ or $\overline{\text{MS1}}$ is asserted whenever the TigerSHARC processor accesses memory banks 0 or 1 respectively. $\overline{\text{MS1-0}}$ are decoded memory address pins changing concurrently with address pins during memory accesses.</p> <p>In multiprocessing systems, $\overline{\text{MS1-0}}$ are driven by the master.</p>
$\overline{\text{MSH}}$	<p>Memory Select Host</p> <p>$\overline{\text{MSH}}$ is asserted whenever the TigerSHARC processor accesses the host address space. The $\overline{\text{MSH}}$ pin is a decoded memory address pin changing concurrently with address pins. In a multiprocessing system, $\overline{\text{MSH}}$ is driven by the master.</p>
$\overline{\text{FLYBY}}$	<p>Flyby</p> <p>When a TigerSHARC processor DMA channel is initiated in flyby mode, it generates flyby transactions on the external bus. During flyby transactions, the $\overline{\text{FLYBY}}$ pin is asserted, indicating the source or destination I/O device where the next data should be placed, or where to strobe the current data. This event also signals that the system should be ready for the next data on the next cycle.</p>
$\overline{\text{IOEN}}$	<p>I/O device Output Enable</p> <p>$\overline{\text{IOEN}}$ enables the output buffer of I/O device in flyby transactions from I/O to Memory. Active on flyby transactions.</p>
$\overline{\text{BRST}}$	<p>Burst</p> <p>$\overline{\text{BRST}}$ is asserted by the current bus master (TigerSHARC processor or a host) to indicate data associated with consecutive addresses is being read or written. A slave device may ignore addresses after the first one and increment an internal address counter after each transfer. If the burst access is from the host to the TigerSHARC processor, the TigerSHARC processor increments the address automatically as long as $\overline{\text{BRST}}$ is asserted.</p>

where the arbitration between multiple TigerSHARC processors and between TigerSHARC processors and the host is accomplished using a distributed arbitration logic on the TigerSHARC processors themselves.

Processor Microarchitecture

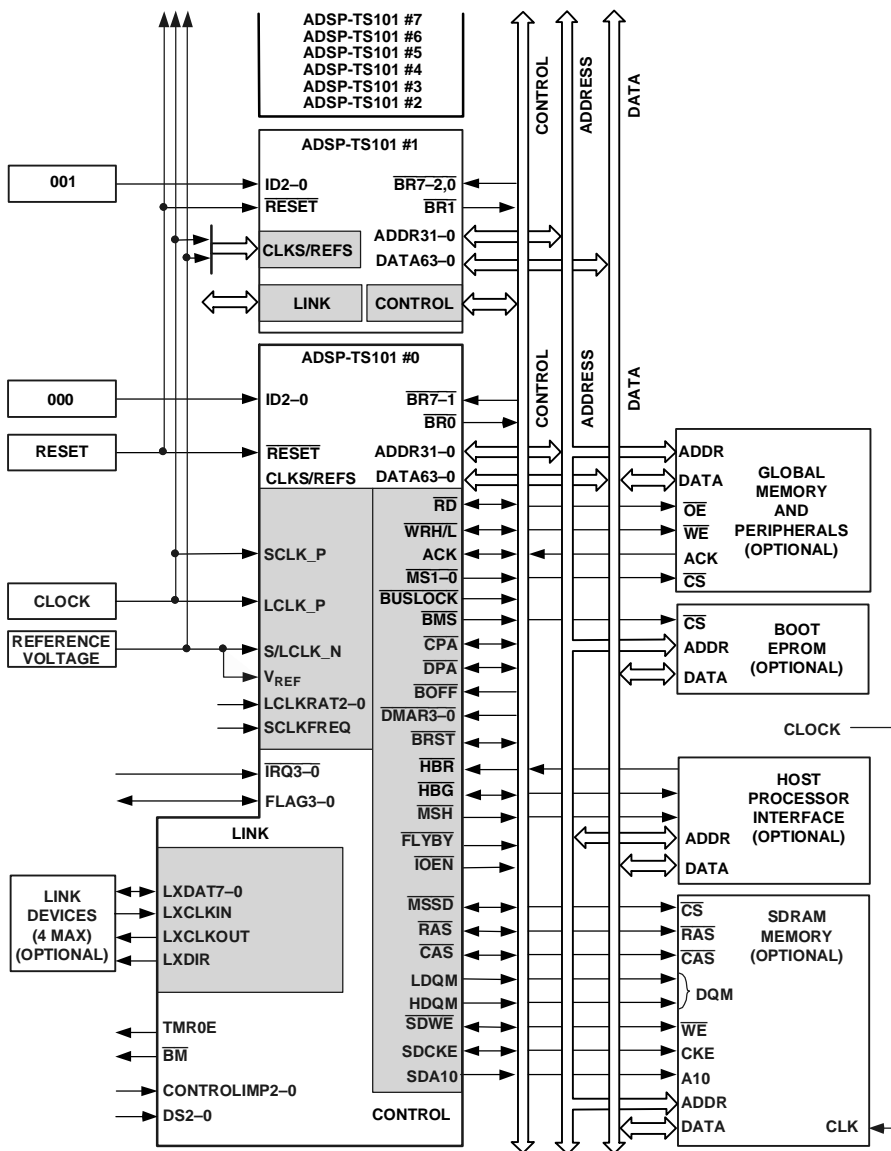


Figure 5-1. Typical Multiprocessing Cluster Configuration

In a multiprocessor system, each TigerSHARC processor must be identified by its own ID. The ID is set by three input pins that are connected to a constant value. Single processor ID must always be zero. Also, in a system where multiple TigerSHARC processors are present, the ID of one processor must be zero. For more information, see “Multiprocessing” on page 5-38.

Other agents defined on the external bus can be memory chips, I/O devices, or bridges to other buses. Additionally, every TigerSHARC processor and the host in the system can be accessed as slaves.

Transaction on the external bus can be word (32-bits), long (64-bits) or quad (128-bits) length. The address must be aligned to its size—that is, quad-word transactions must be quad-word aligned and long-word must be long-word aligned. Since the bus width (32- or 64-bits) may be smaller than the transaction size, a transaction may be spread over a few cycles in order to be transferred in its entirety.

External Port Input FIFO (IFIFO) refers to the TigerSHARC processor Bus Interface Unit (BIU) input FIFO. It is used for all externally-supplied data by bus masters (other TigerSHARC processors or a host processor), direct writes or external reads. The IFIFO also holds on-chip destination addresses and data attributes.

External Port Output FIFO (OFIFO) refers to the TigerSHARC processor BIU output FIFO. It is used for all outgoing external port addresses, data, and transaction control signals, including DMA transfers to and from external address space.

The source of the transaction can be either the core (instruction fetch or operand access) or the DMA. In order to optimize the transactions flow, it is important to have a general understanding of the BIU. The external port architecture is illustrated in Figure 5-2 on page 5-8. An external transaction is initiated by the internal master (DMA or core) on one of the three internal buses. The transaction is strobed by the OFIFO and, in turn, is driven on the external bus. If the transaction is a write transaction,

Processor Microarchitecture

the transaction is completed at this point. If it is a read transaction, the data read on the external transaction is strobed in the IFIFO and is written later to its target (defined by the original internal transaction).

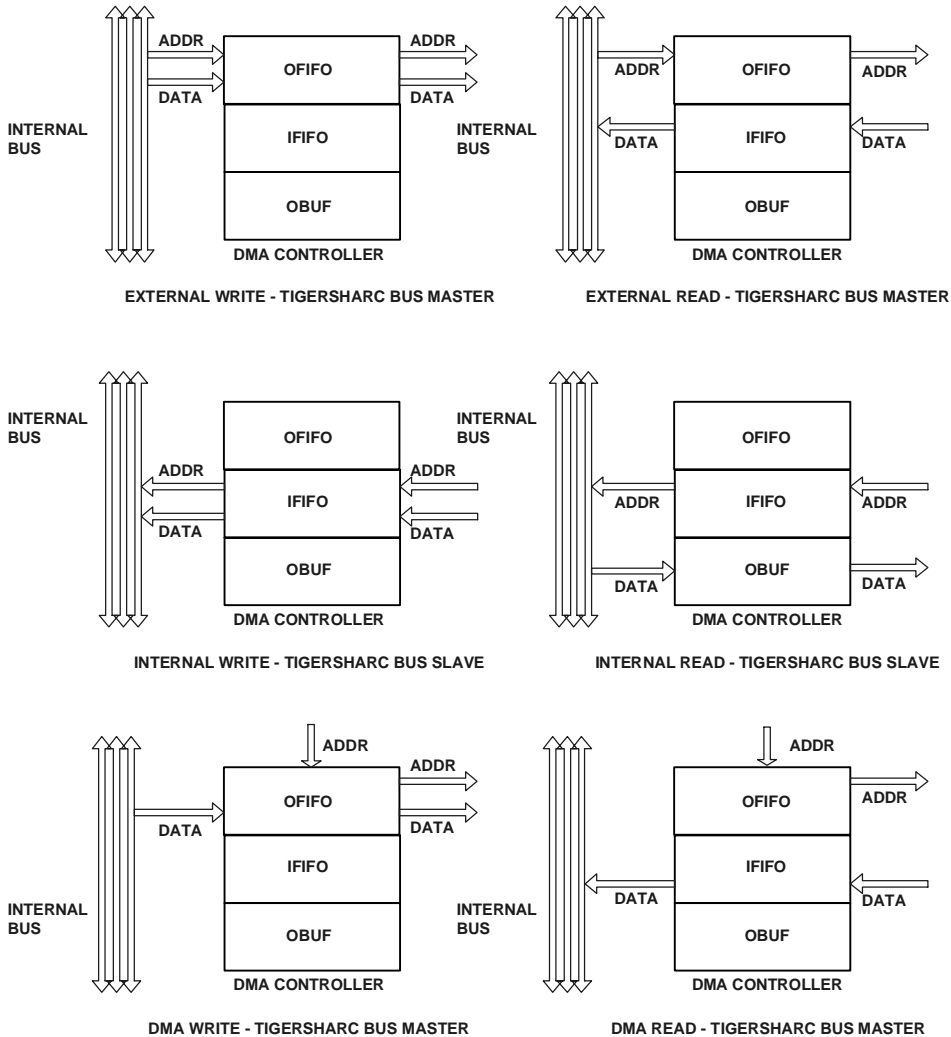


Figure 5-2. External Port Architecture

Transactions can be initiated on the external bus and driven into internal TigerSHARC processor elements. For example, memory and Uregs can be read and written by an external bus master—another TigerSHARC processor or host. The transaction is identified according to the memory map (see “Multiprocessor Space” on page 2-5) and processor ID, strobed by the IFIFO, and driven on the internal bus in turn. If the external transaction is read, the data is returned to the master that initiated the transaction through the slave’s OBUF.

On DMA transactions (reads or writes), addresses are written to the OFIFO through the I/O address bus and not the internal address bus in order to reduce internal bus activity. Only the data read from the external bus is transferred on an internal bus.

The internal buses are used for both external and internal transactions, where each bus can execute a new transaction every cycle. The cause of a transaction can be instruction fetch, DMA transfer, Load or Store instruction or IFIFO (internal address accessed by an external master or data returned from an external read).

The bus selection for the execution of a particular transaction is performed according to memory address. If internal memory is involved in the transaction, the selected bus is the bus of that memory. (For example, if the access is targeted to memory block 0, the transaction is executed on bus #0.)

When a transaction occurs between registers or between external memory and a register, it can be executed on any internal bus. This type of transaction, called a “virtual bus transaction”, is executed on the first available internal bus. If no bus becomes available for four cycles, one of the three internal buses is forced to service the virtual bus transaction. Even when more than one bus is not assigned to regular bus transactions, only one virtual bus transaction can be executed in a given cycle.

Processor Microarchitecture

Arbitration is performed on each internal bus separately, including the virtual bus. The priority on the internal bus is:

- High priority IFIFO transactions
- High priority DMA transactions
- Load, Store, and other data transfer instructions
- Low priority IFIFO transactions
- Low priority DMA transactions
- Instruction fetch

The DMA request priority is determined by the source TCB priority bit (see “DPx Register” on page 7-18).

The IFIFO request priority is high in the following cases.

- Direct read by an external master
- Broadcast write transaction in the IFIFO
- Write to internal address (the result of a DMA transaction) and the destination TCB priority bit is set
- IFIFO is full (three or more transactions in the IFIFO). The request priority is high in order to prevent delays on the external bus.

In all other cases, the IFIFO request has low priority.

SYSCON Programming

The SYSCON register, described in “” on page 2-34, is the system configuration register and must be programmed before bus traffic begins. This register may be programmed only once after reset, where any further writes to the SYSCON register after the first programming are ignored.

For each of the three banks there are six mode bits—the fields are shown in Figure 5-3. The setup of these fields is orthogonal—each of the fields can be programmed to a different value.

SYSCON Programming

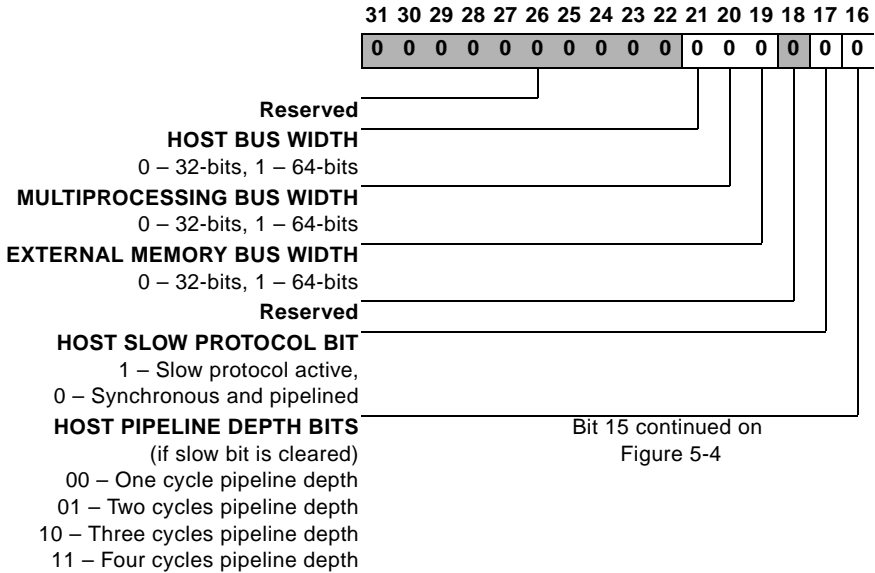


Figure 5-3. SYSCON (Upper) Register Bit Descriptions

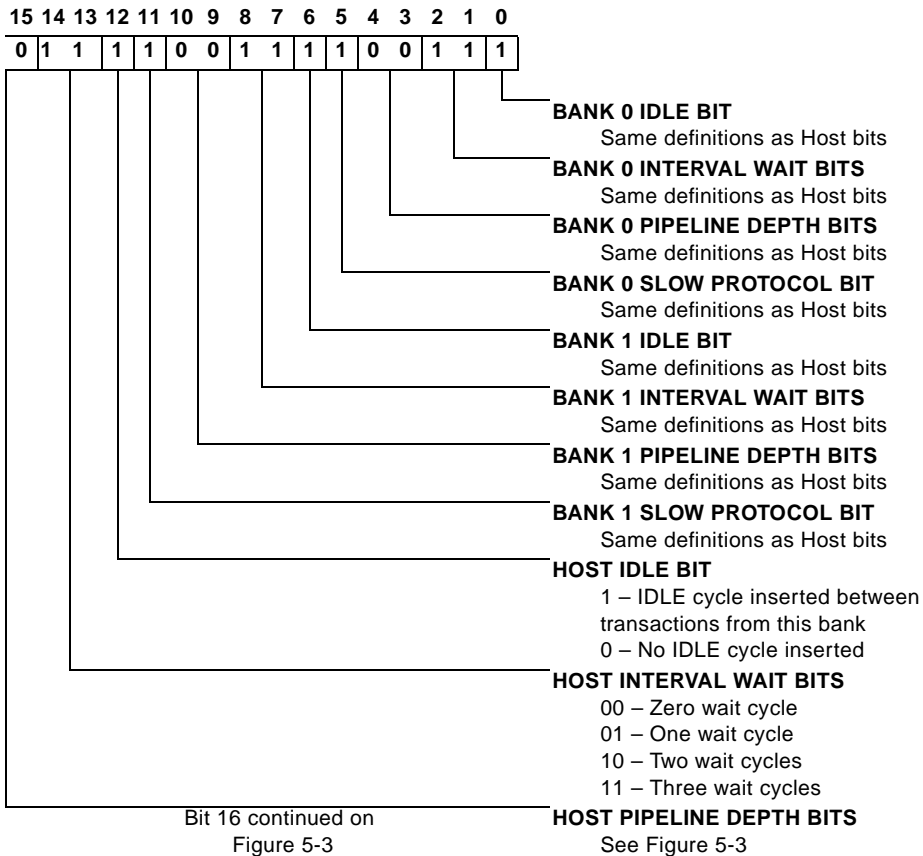


Figure 5-4. SYSCON (Lower) Register Bit Descriptions

Bus Width

The bus width is configured separately for memory access, host transactions (either as master or as slave), and multiprocessing. The setup is zero for a 32-bit bus and one for a 64-bit bus. Note that if either the host or memory bus width is 64-bits, the multiprocessing width must also be 64-bits. The valid width settings are shown in Table 5-2.

Table 5-2. Valid Width Settings

000	allowed
001	not allowed
010	allowed
011	allowed
100	not allowed
101	not allowed
110	allowed
111	allowed

Slow Device Protocol

For slow device protocol setup, the slow protocol bit (Bit5 for bank 0, Bit11 for bank 1, and Bit17 for host) is set. Additionally, the pipeline depth field is set to 0b00 and the IDLE bit is a “don’t care” bit—it can be programmed either way. The internal wait field identifies the number of internal wait cycles on slow accesses. If the number of internal waits is zero, the external wait mechanism cannot be used for these transactions.

Pipelined Protocol

For pipelined protocol setup, the slow protocol bit is cleared and the pipeline depth field is set to the required pipeline depth (0b00 for one cycle, 0b01 for two cycles, 0b10 for three cycles, and 0b11 for four cycles). This defines the pipeline depth for read transactions only; pipeline depth for write transactions is always one. The IDLE bit is set if there are multiple slaves in the address range of this bank, in order to prevent contention on the data bus on transitions between different slaves on consecutive reads. If this bank contains a single slave, the IDLE bit should be cleared. The internal wait field is irrelevant for pipelined transactions. In pipelined protocol, there is no internal wait state programming.

Initial Value

The initial value after reset, which can be used as the default, is as follows.

- Bus width: 32 for host, multiprocessing and memory access
- Bus configuration: slow protocol with three wait states

TigerSHARC Pipelined Interface

The TigerSHARC processor uses the pipelined protocol to interface with other TigerSHARC processors, the host, and fast synchronous memories. The transaction is also performed by the pipelined protocol when the TigerSHARC processor is accessed as a slave.

This protocol was created for pipelining the transactions with a throughput of one datum per cycle, although the latency of the transaction can be up to four cycles. The address and controls of a transaction are issued on the address cycle and the data is transferred a few cycles later—in the case of TigerSHARC processor, one to four cycles depending on the transaction direction and system configuration programming. The TigerSHARC

TigerSHARC Pipelined Interface

processor can issue an address of a new transaction every cycle and doesn't need to wait for the data cycle of the first transaction before beginning the address cycle of the new transaction.

Control Signals

The control signals used in pipelined transactions are listed below.

- \overline{RD} – the transaction is read.
- \overline{WRH} and \overline{WRL} – the transaction is written if one of these is active. The two signals also indicate which word on the data bus is valid.
- \overline{BRST} – Indicates the next cycle belongs to the same transaction as the current cycle—for example, when accessing a quad-word on a bus width of 32-bits, \overline{BRST} is active on the first three cycles of the four cycle transaction.
- ACK – Driven by the target slave on the data cycle. If asserted, the slave is ready to complete the data cycle; otherwise wait cycles are generated.

Basic Transaction

The basic pipelined transaction is shown in Figure 5-5 on page 5-17. In the address cycle, the address is issued with the \overline{RD} or \overline{WRX} (\overline{WRL} , \overline{WRH} or both) asserted. The data cycle begins after one to four cycles, as specified in the target slave configuration and the direction of the transaction. The delay between the address cycle and the data cycle is the “pipeline depth”. In the data cycle, the data is transferred according to the transaction direction. If the slave is ready, it asserts the ACK signal and either drives or strobes the data. If the slave is not ready, it deasserts the ACK signal and delays the data. The exact way the ACK signal behaves is discussed in “Wait Cycles” on page 5-23.

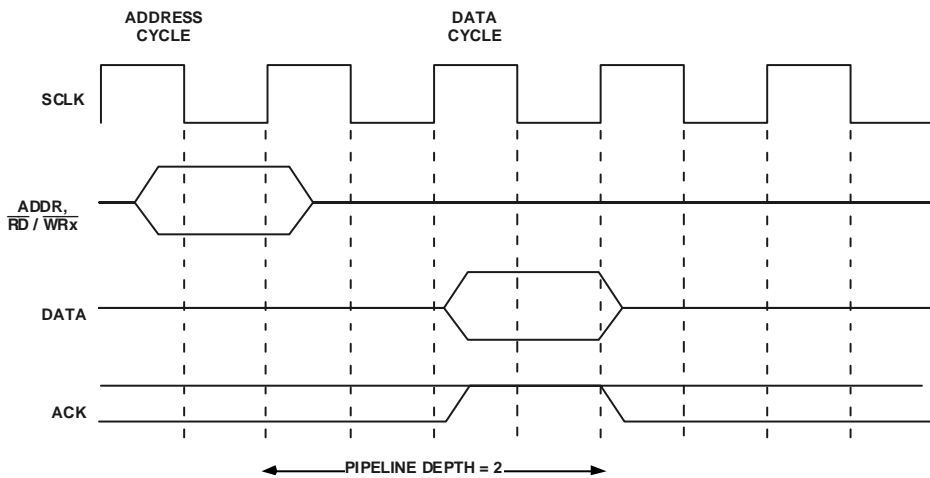


Figure 5-5. Basic Pipelined Transaction

The pipeline depth is configured according to the slave and the transaction type (read or write). For write transactions, the pipeline depth is always 1. For read from another TigerSHARC processor in multiprocessing systems, the pipeline depth is always 4. When reading from external memory banks or host memory space, the pipeline depth is user-programmable and can be up to four cycles. Pipeline depth can be selected individually for each of the banks (bank0, bank1, and host) in the SYSCON register after reset.

Single transactions take a different number of cycles according to transaction size and external bus width:

- Single transactions: one cycle
- Long transactions on 64-bit bus: one cycle
- Long transactions on 32-bit bus: two cycles

TigerSHARC Pipelined Interface

- Quad transactions on 64-bit bus: two cycles
- Quad transactions on 32-bit bus: four cycles

The transaction type is determined by the \overline{RD} and $\overline{WRL}/\overline{WRH}$ signals on the address cycle. The \overline{WRL} and \overline{WRH} signals also determine which words of the 64-bit bus should be written to memory. See the External Port Alignment Figure 6-3 on page 6-12.

Pipelining Transactions

The pipeline is effective when used in sequential accesses having the same pipeline depth. Sequential accesses may be the result of one transaction which is too wide for the bus width, or merely distinct pipelined transactions. The first is called “burst transaction” and is indicated by the \overline{BRST} pin. The \overline{BRST} pin is valid on the address cycle.

The \overline{BRST} signal is used to indicate transaction continuation in pipelined protocol transactions. It is used to qualify consecutive accesses on the system bus and indicates that the next cycle belongs to the same transaction as the current cycle.

The \overline{BRST} signal is important to the TigerSHARC processor when it is acting as a slave. The \overline{BRST} signal indicates that multiple transactions that are part of a burst access are to be interpreted as a single transaction. One example can be illustrated in a multiprocessor system where one TigerSHARC master attempts to access another TigerSHARC slave's TCB register. TCB registers may only be accessed via quad-word accesses and a multiword transfer over the system bus must be qualified by the \overline{BRST} signal so that it can be properly interpreted as a quad-word access by the slave TigerSHARC. Without the \overline{BRST} signal, this same access would be interpreted as four single-word (i.e., 32-bit normal word) accesses which would result in an illegal TCB register access.

A generic slave device (i.e., a device other than a TigerSHARC acting as slave) can use the $\overline{\text{BRST}}$ signal to accept the first address in a burst transfer and then automatically increment that address as successive data words arrive.

The $\overline{\text{BRST}}$ signal must be connected in a multiprocessor TigerSHARC system in order to qualify long-word and quad-word accesses between DSPs. The $\overline{\text{BRST}}$ signal may optionally be connected between a host device and a TigerSHARC. If burst accesses (including but not limited to long-word and/or quad-word accesses) between host and TigerSHARC processor are required, then $\overline{\text{BRST}}$ must be connected and used to qualify these accesses for the TigerSHARC slave.

Figure 5-6 on page 5-20 and Figure 5-7 on page 5-21 illustrate examples of different types of sequential transactions in a 32-bit bus. Figure 5-6 on page 5-20 shows a quad-word write that is executed in four cycles, where the $\overline{\text{BRST}}$ pin is asserted on the first three cycles. This transaction is followed by a word write, and then two long writes where the $\overline{\text{BRST}}$ is asserted in the first cycle of each transaction. The pipeline depth is one cycle since these are write transactions.

TigerSHARC Pipelined Interface

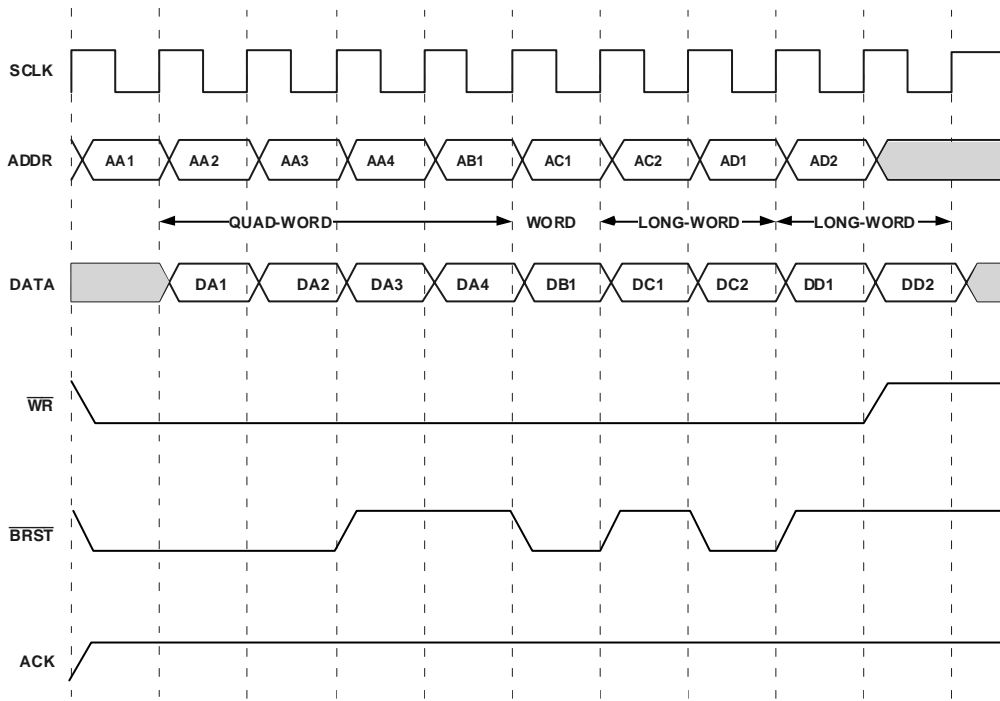


Figure 5-6. Pipelined Transactions – Sequential Writes
Pipeline Depth = 1, Bus Width = 32

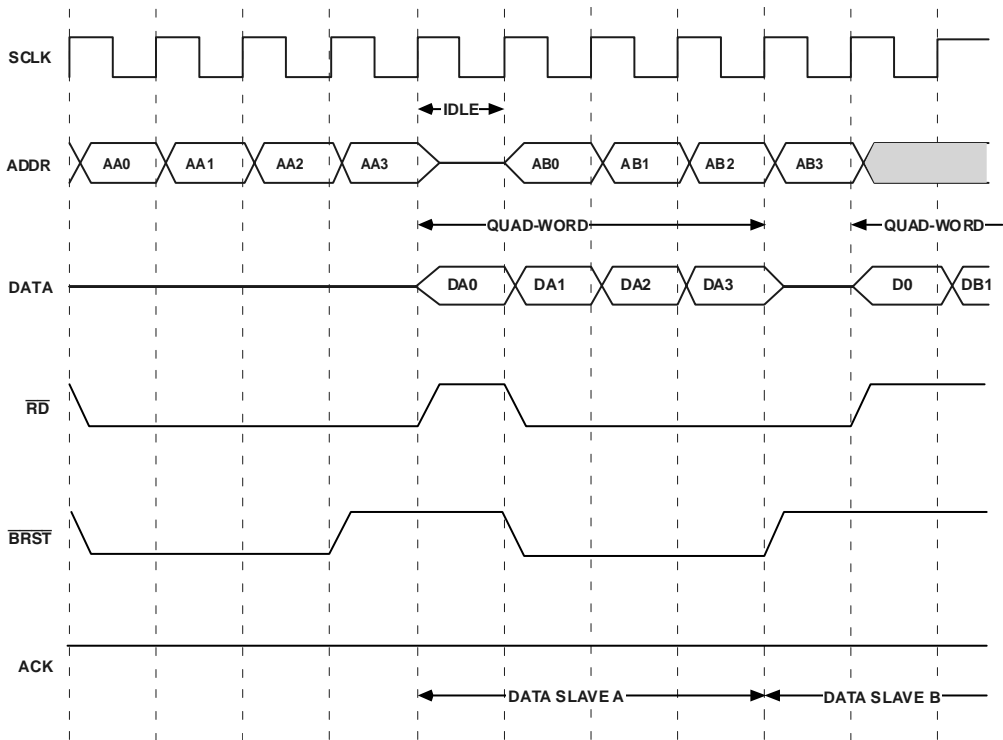


Figure 5-7. Burst Read Followed by Burst Read
 Pipeline Depth = 4, Bus Width = 32, IDLE Bit Set

TigerSHARC Pipelined Interface

An IDLE cycle is inserted between transactions to prevent contention between drivers. Figure 5-7 on page 5-21 shows two quad-word read transactions where each transaction takes four bus cycles. The two transactions are separated by an IDLE cycle. IDLE cycles are inserted in the following cases.

- Read followed by read from two different zones, different TigerSHARC processors, host or combination of the above, if the second read's pipeline depth is smaller or equal to the first read pipeline depth.
- Read followed by read from the same zone, and the IDLE bit of this zone in the SYSCON register is set.
- Read followed by write or write followed by read, if the second transaction's pipeline depth is smaller or equal to the first transaction pipeline depth.

If the corresponding IDLE bit is cleared, sequences of write, a sequence of a multicycle transaction, or sequences of reads from the same slave are not separated with an IDLE cycle.

Sequential accesses with different pipeline depths receive special treatment. If the first transaction has a smaller pipeline depth than the second transaction, the address cycles are issued with no gap, even though there may be a gap between the data cycles. If the pipeline depth of the first transaction is larger than the pipeline depth of the second, the transactions cannot be sequential. As shown in Figure 5-8 on page 5-23, the second transaction address cycle is delayed after the first transaction. The purpose of the delay is to keep the data cycles of the two transactions separate. The delay between the two transactions is the difference in the pipeline depth if no IDLE cycle is inserted, or the difference between the pipeline depths plus one if the conditions for IDLE are true. If the difference between the pipeline depths is one, an IDLE cycle is inserted. The second transaction address cycle begins two cycles after the end of the first transaction.

The illustrated sequence in Figure 5-8 assumes a different pipeline depth and a bus width of 32.

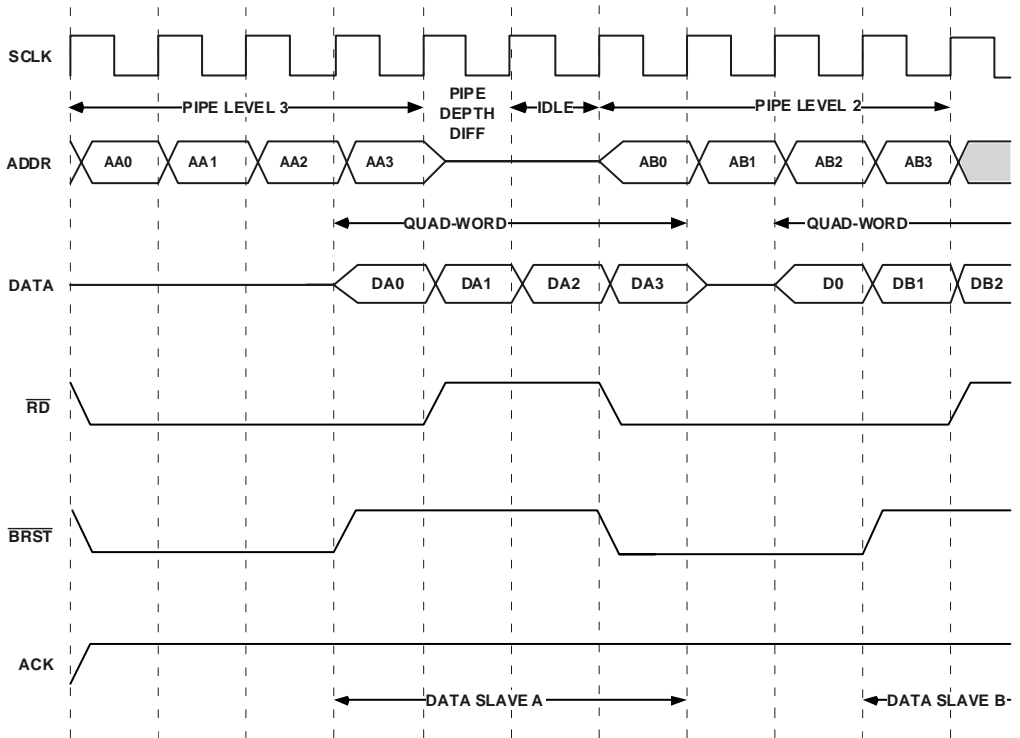


Figure 5-8. Read Followed by Read From Different Slave

Wait Cycles


In regular transactions, if the slave is ready in time for the data cycle of the targeted transactions, the slave asserts the **ACK** signal in the data cycle. If the slave is not ready, it deasserts the **ACK** signal on the data cycle and keeps it deasserted until it can continue.

TigerSHARC Pipelined Interface

The `ACK` signal is valid on the data cycle and is driven by the slave that is in the data cycle. When no slave drives the data cycle, it must be driven high by an external pull-up on the `ACK` signal. The `ACK` signal can be issued in response to any data cycle in a multicycle transaction. There is no restriction on its length.

When the `ACK` signal is deasserted, it stalls all the outstanding transactions, including transactions that are targeted to other slaves. Since the bus is stalled on the cycle following the active `ACK` cycle, the slave has to sample any new address that the master may issue on the cycle that the `ACK` signal was deasserted. Figure 5-9 on page 5-25 shows an example of `ACK` signal usage. In this example, the datum, `da1`, is not ready in time and is issued one cycle later. The `ACK` signal is deasserted on the `da1` data cycle and asserted in the next cycle to indicate valid data. The consequences of the wait cycle are listed below.

- The master samples the data one cycle later.
- The address cycle following the `ACK` signal (`ab2`) is stretched by one cycle.
- All the slaves that have pending transactions on the cycle following the `ACK` signal are stalled for the number of cycles that the `ACK` signal was deasserted. For example, between the `ab0` address cycle and the `db0` data cycle there are five cycles, although the pipeline depth is only four.

 The address and controls driven on the bus on the cycle of the `ACK` signal must be latched by the slave.

The sequence illustrated in Figure 5-9 is true when the `ACK` signal is not active and Bus Width = 32.

Write transactions use the `ACK` signal wait cycles differently from read transactions. Here the slave asserts the `ACK` signal as long as its write buffer has sufficient free slots. See Figure 5-10 on page 5-26. In this example, the address cycle, `aa3`, triggers the slave to deassert the `ACK` signal. This

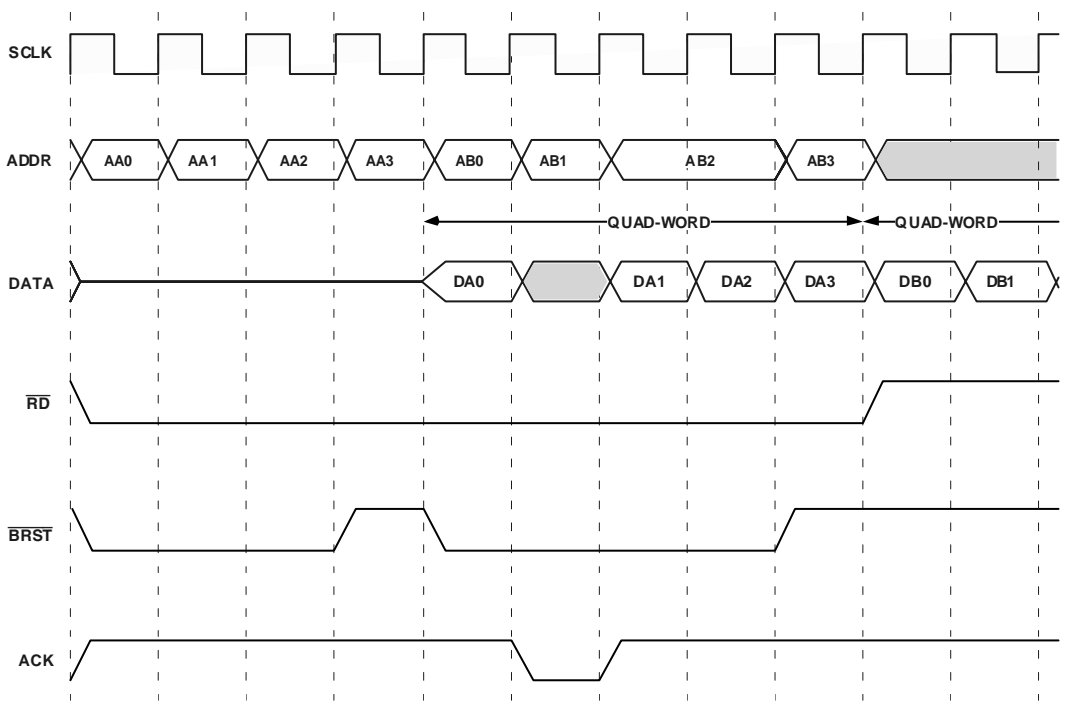


Figure 5-9. Burst Read Access Followed by Burst Read

extends the transaction pipeline for two cycles. The extended cycles begin one cycle after the ACK signal was deasserted—for example, ab2 and db1 are extended for two additional cycles. All the slaves strobe address ab1, and the target slave of transaction samples datum, db0, in the cycle that the ACK signal is deasserted.

- i In Figure 5-10 on page 5-26, transaction ab0 and ab1 are targeted to different slaves. If a slave drives the ACK signal low and the following transaction is to a different slave, the target slave of the second transaction may not drive the ACK signal before the first slave has completed the transaction. This applies to the cycle fol-

TigerSHARC Pipelined Interface

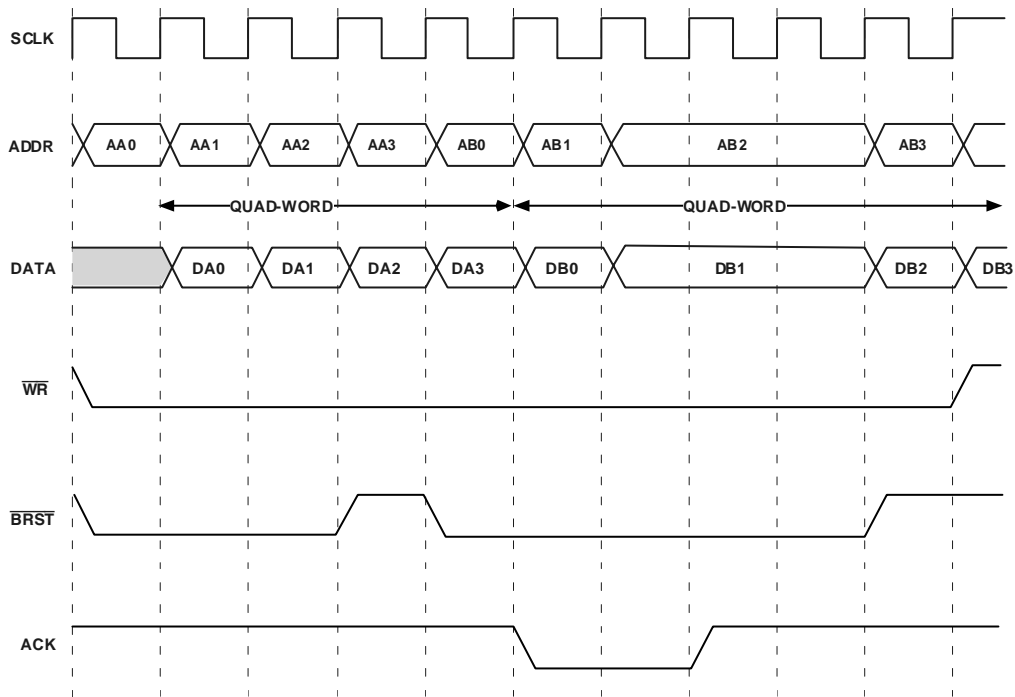


Figure 5-10. Burst Write Access Followed by a Burst Write

lowing the cycle in which the ACK signal was high. If this rule is violated, there may be contention on the ACK signal which can be solved in one of three ways:

- Make all slaves work immediately at the ACK signal on write. (FIFOs should be cleared faster than being written to.) You may use the fact that the ACK signal has a pull-up resistor that keeps it set when not driven.
- Centralized logic for the ACK signal
- Keep the ACK signal logic in slave very fast—enable drive of the ACK signal only after the ACK signal has been driven high in the previous cycle.

Slow Device Protocol

The TigerSHARC processor supports a slow device protocol that can be used for simple devices. The slow device protocol can be configured for address spaces belonging to bank0, bank1, or host. The slow device protocol is set by programming the relevant bits in the `SYSCON` register.

- Slow = 1
- Pipeline depth = 0b00 (fixed when slow bit is set)
- Wait cycles = programmed according to system requirements
- IDLE cycle = 1 (fixed when slow bit is set)

The purpose of this setup is to enable a direct connection to simple and low-performance, non-critical memories or peripherals. The protocol can work with synchronous or asynchronous devices.

The basic protocol, when the configuration is “zero wait cycles”, is shown in Figure 5-11 on page 5-28 and Figure 5-12 on page 5-29. The memory select is asserted to begin the transaction and the address is driven with it. In the next cycle, the \overline{RD} or \overline{WRX} signal (according to the transaction type) is asserted. Data is driven by the TigerSHARC processor if the transaction is write. If the transaction is read, the slave can start driving the data. The TigerSHARC processor latches the data at the end of this cycle. For the zero-wait-cycles configuration, wait cycles cannot be inserted by deasserting the `ACK` signal.

In the slow protocol, the $\overline{MS0-1}$ signals act as control signals and not as select signals. The external memory device used with this protocol should be able to latch the data on the first rising edge of either $\overline{MS0-1}$ or the \overline{WR}

Slow Device Protocol

signal. One of the key requirements for this device is that it meets the hold time for both address and data. The right number of wait cycles to guarantee enough data setup time is user configurable.

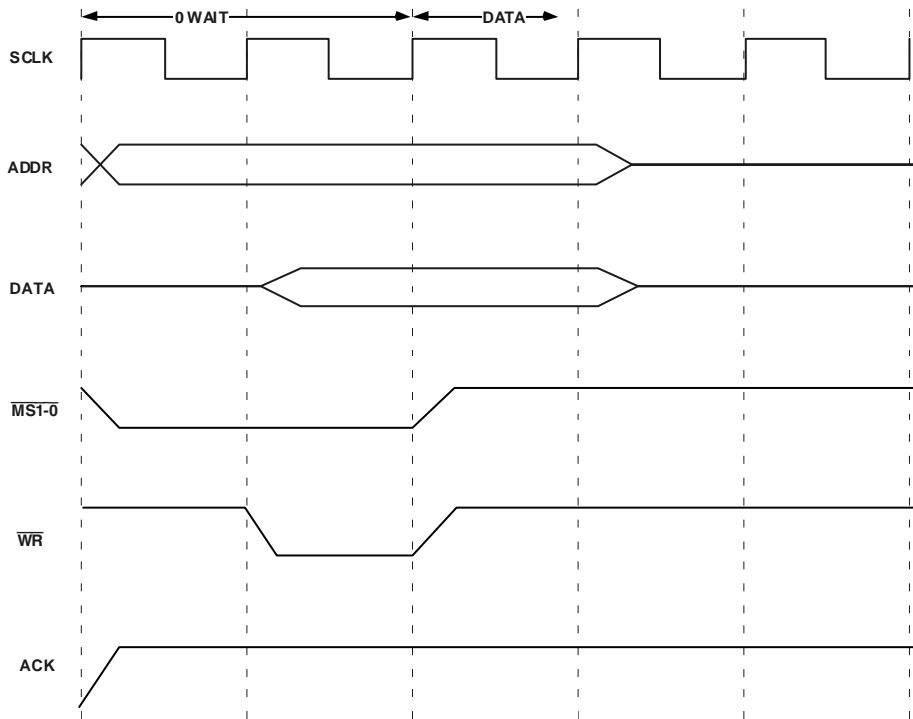


Figure 5-11. Slow Protocol Write With 0 Wait Cycles

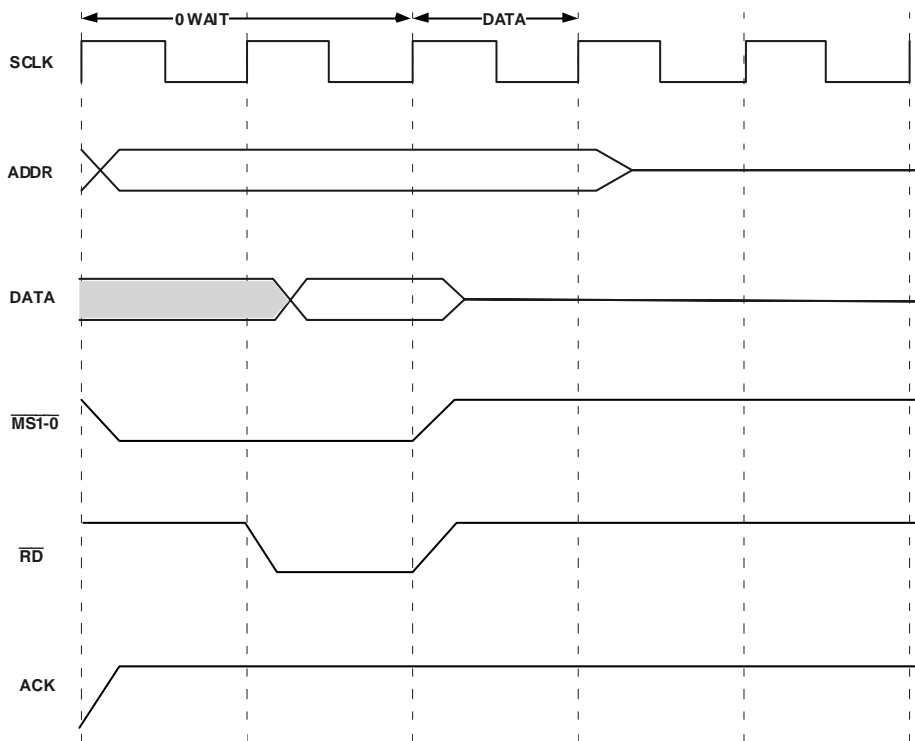


Figure 5-12. Slow Protocol Read With 0 Wait Cycles

Slow Device Protocol

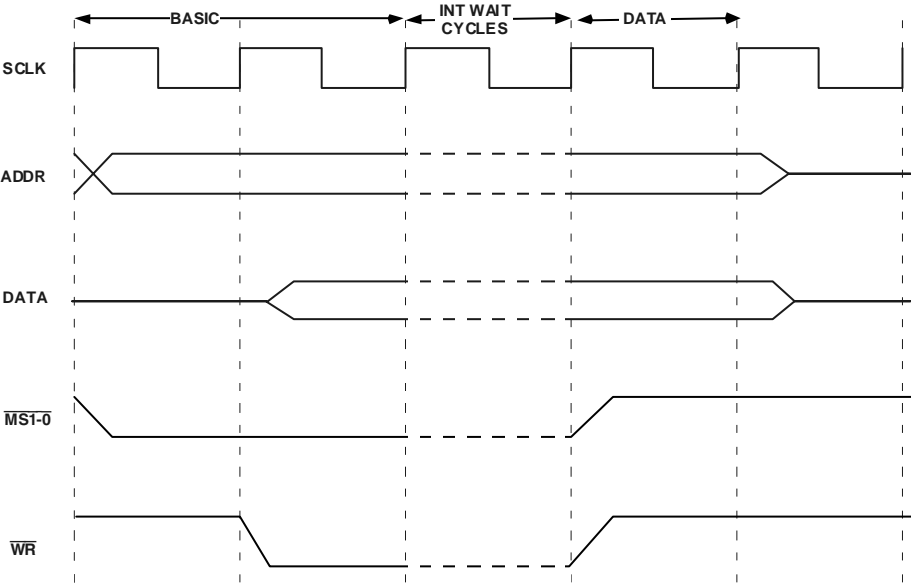


Figure 5-13. Slow Protocol Write With Wait Cycles

In slow device protocol, wait cycles can be inserted as required. The `SYSCON` register includes a field for the number of internal wait cycles—between zero and three. Figure 5-13 on page 5-30 and Figure 5-14 on page 5-31 illustrate slow transactions with one or more wait cycles. This situation is similar to the zero-wait-state transaction, but the second cycle (one cycle after the memory select is asserted) is repeated according to the number of internal wait cycles.

External wait cycles can be inserted by deasserting the `ACK` signal. This can only be done when at least one internal wait state has been configured. To insert external wait cycles, the `ACK` signal must be deasserted one cycle or more before the last wait cycle. The extra wait cycles are repeated until one cycle after the `ACK` signal has been asserted. See Figure 5-15 on page 5-32 for an example.

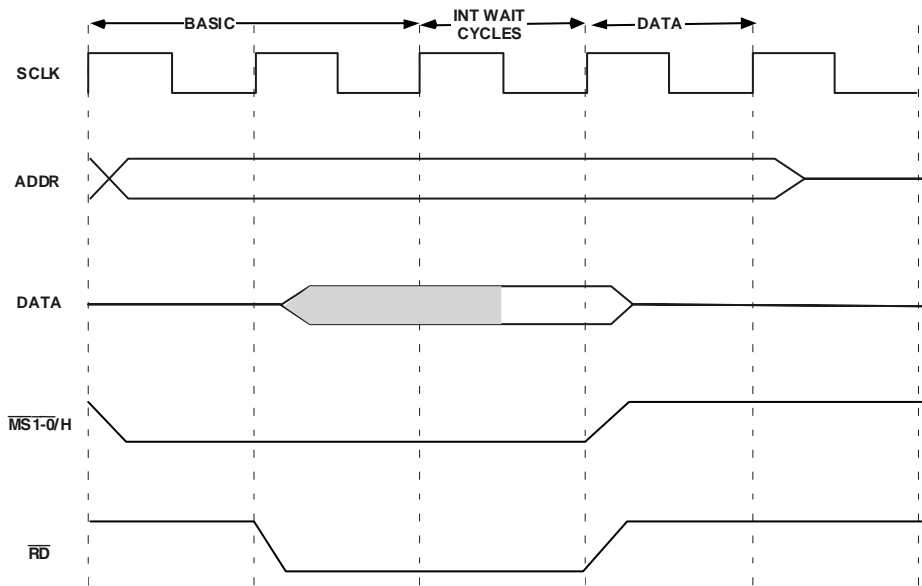


Figure 5-14. Slow Protocol Read With Wait Cycles

EPROM Interface

The TigerSHARC processor can be configured during reset to boot from an external 8-bit EPROM. In this case, the program is loaded from the EPROM into internal memory by an automatic process as part of the reset sequence. The TigerSHARC processor uses a specific DMA channel to load the program. The TigerSHARC processor bus interface packs bytes to 32-bit instructions. EPROM may also be accessed during normal work via DMA. The boot EPROM cannot be accessed by the core. The EPROM is a byte address space and is not part of the TigerSHARC processor memory space. It is limited to 16M bytes (maximum address is $0xFF\ FFFF$, $ADDR_{31-24} = 0$). The data is driven on the regular data bus Bits7–0.

EPROM Interface

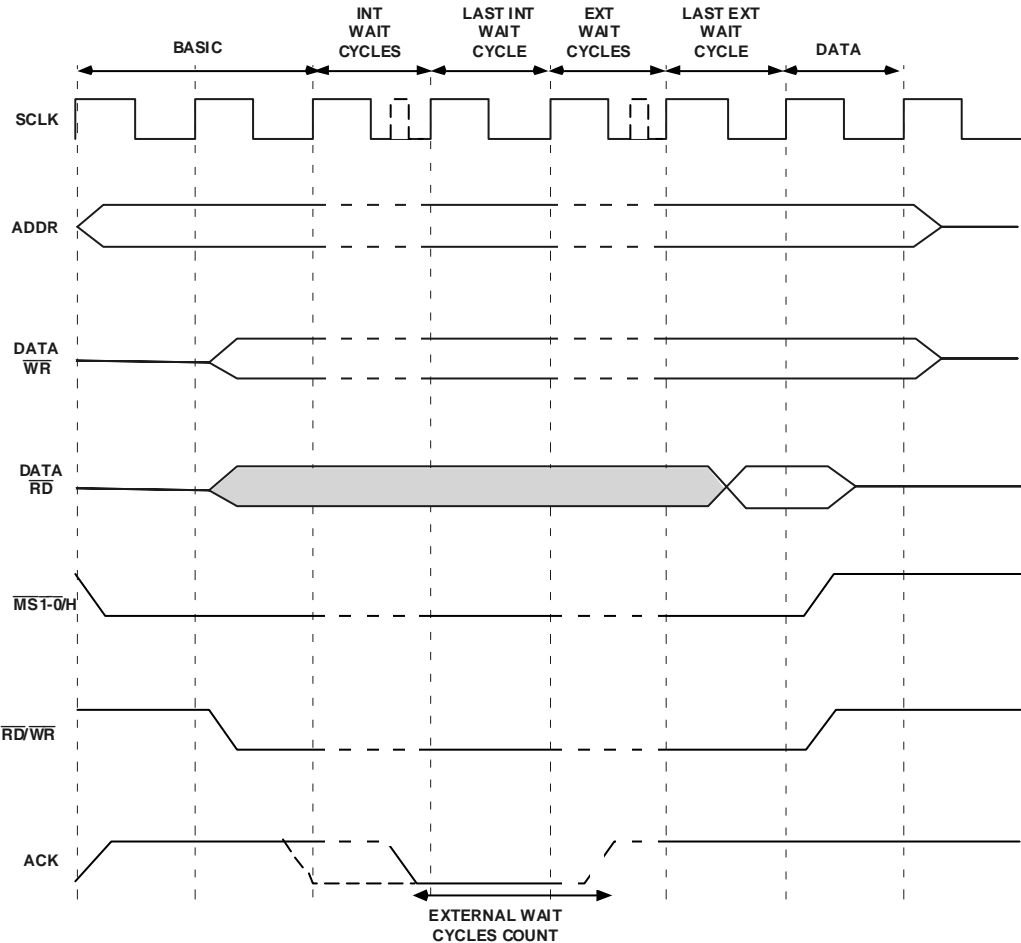


Figure 5-15. Slow Protocol Read/Write With External Wait Cycles

The TigerSHARC processor uses 16 wait cycles for each read access from the EPROM. During the boot process, the $\overline{\text{BMS}}$ pin is used as the EPROM CS pin until completion.

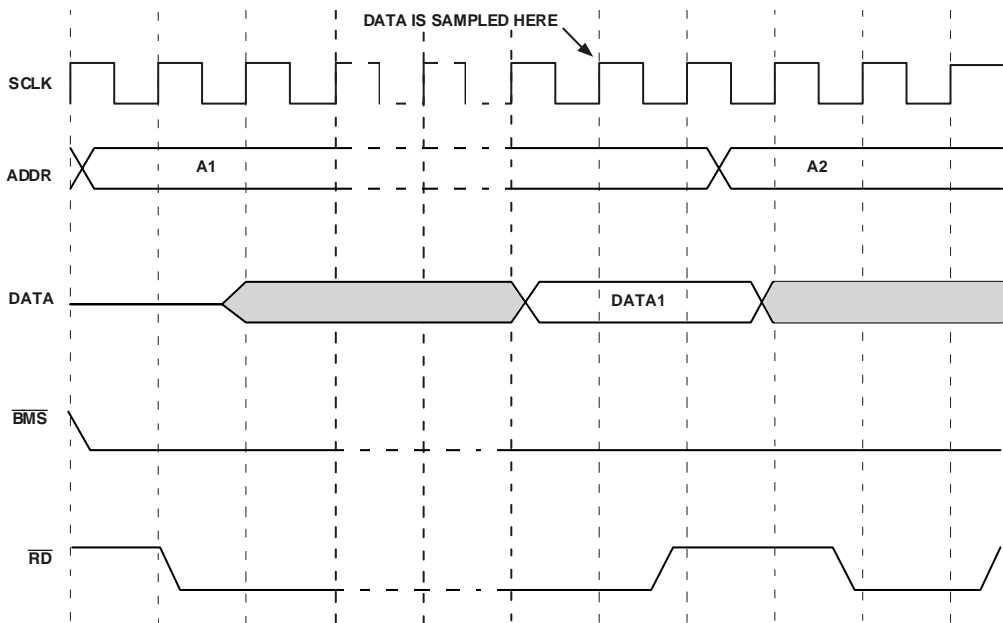


Figure 5-16. Access to Boot EPROM – 16 Wait Cycles

This type of transaction can be operated by the DMA channels anytime. The transaction can also operate as write for flash EPROM programming. The write transaction is identical to the EPROM read, but the write does not scatter the bytes. When a word is written to the flash memory, it is driven on the external bus on Bits31–0. Since the flash memory is connected only to the bottom byte, the useful data is only on Bits7–0. It is the programmer's responsibility to distribute the data on the bottom bytes of the words. In TigerSHARC processor's internal or external RAM, the data should be organized as shown in Figure 5-17. After the last boot EPROM bus cycle, there are three IDLE cycles (for slow EPROM disconnect time).

Flyby Transactions

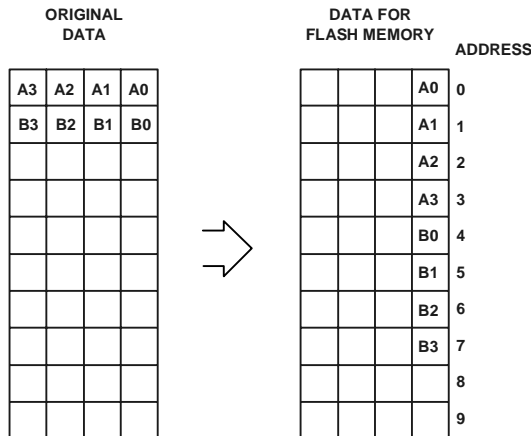


Figure 5-17. Data Preparation for Write to 8-Bit FLASH

Flyby Transactions

Flyby transactions are used by the DMA for transferring data between an external I/O device and external memory (for example, DMA may be programmed to transfer data from an A/D device to the SDRAM). A flyby transaction is issued by a DMA channel that is programmed to execute flyby transactions (see “DPx Register” on page 7-18). The flyby transaction can be any type of transaction—pipelined, slow, or SDRAM.

A flyby transaction type can be from memory to I/O (Flyby Read) or from I/O to memory (Flyby Write). In flyby transactions the TigerSHARC processor relinquishes the data bus during the transaction and then activates in the external memory and the I/O devices in parallel. Memory is driven in the regular way and the I/O device is controlled with two pins, $\overline{\text{FLYBY}}$ and $\overline{\text{TOEN}}$.

The $\overline{\text{FLYBY}}$ pin is active in every data cycle of a flyby transaction. For read transactions, the $\overline{\text{FLYBY}}$ pin is an indication to the I/O device to strobe the data on the data bus and to increment its internal state machines or FIFOs, if necessary. In write transactions, this is slightly more complicated. The $\overline{\text{FLYBY}}$ pin can be used to increment the I/O internal state machines or FIFOs if necessary. It cannot, however, be used for driving data by the I/O because there may be a long delay path from the TigerSHARC processor to the I/O output buffers and from the I/O device to memory—all in one cycle. The I/O output buffers are enabled by the $\overline{\text{IOEN}}$ pin. The $\overline{\text{IOEN}}$ is active at least one cycle before the data cycle, but only after any other device ceases driving the data bus. An example of flyby transactions, in this case a data write to the SDRAM, is shown in Figure 5-18 on page 5-36. The $\overline{\text{FLYBY}}$ pin is asserted on the data cycles da0, da1, db0, and db1. The $\overline{\text{FLYBY}}$ pin is the indication to the I/O device to change the data. The data drive, however, is indicated by asserting the $\overline{\text{IOEN}}$.

The sequence illustrated in Figure 5-18 applies when:

- Transfer is followed by another burst write transfer from I/O to the SDRAM device
- Flyby transaction and bus width = 64

Flyby Transactions

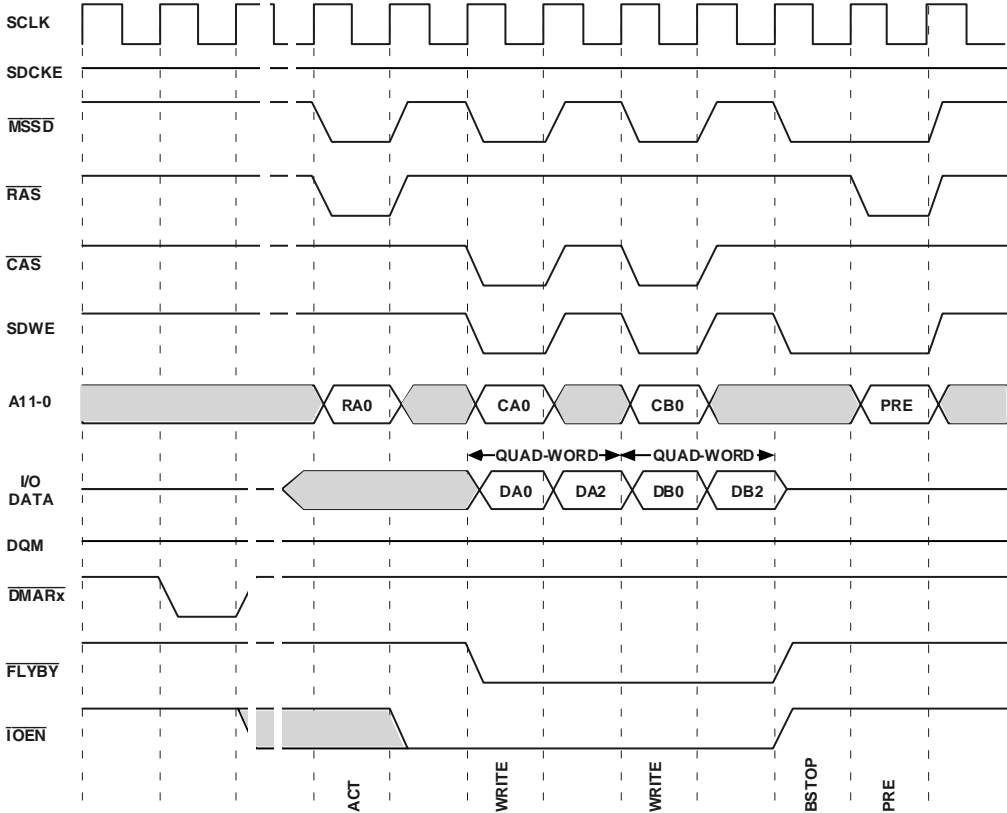


Figure 5-18. Synchronous Burst Write Transfer Followed by Another Burst Write Transfer

The sequence illustrated in Figure 5-19 applies when:

- Transfer is followed by another burst read transfer from the SDRAM to I/O device
- Flyby transaction; $\overline{\text{CAS}}$ latency = 2; bus width = 64

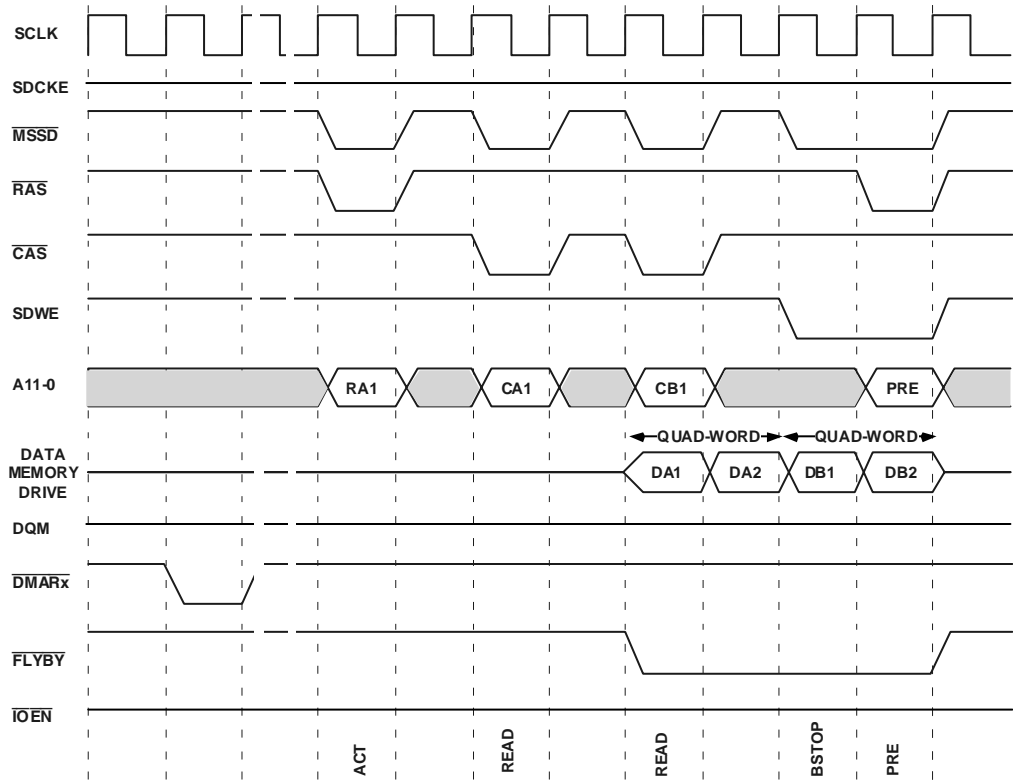


Figure 5-19. Synchronous Burst Read Transfer

Multiprocessing

The TigerSHARC processor incorporates distributed arbitration logic that supports multiprocessing. There can be up to eight processors plus a host connected to the same external bus and each of the bus agents can become the master of the bus.

The I/O pins that are related to multiprocessing are listed in Table 5-3, where type may be Input (I), Output (O), or Three-state (T).

Table 5-3. Multiprocessing I/O Pins

Signal	Description
$\overline{BR7-0}$	<p>Multiprocessing Bus Request Pins</p> <p>Used by the DSPs in a multiprocessor system to arbitrate for bus mastership. Each TigerSHARC processor drives its own \overline{BRx} line (corresponding to the value of its ID2-0 inputs) and monitors all others. In systems with fewer than eight DSPs, set the unused \overline{BRx} pins high.</p>
\overline{HBR}	<p>Host Bus Request</p> <p>A host must assert \overline{HBR} to request control of the TigerSHARC processor external bus. When \overline{HBR} is asserted in a multiprocessing system, the bus master relinquishes the bus and asserts \overline{HBG} once the outstanding transaction is finished.</p>
\overline{HBG}	<p>Host Bus Grant</p> <p>Acknowledges \overline{HBR} and indicates that the host can take control of the external bus. When relinquishing the bus, the master TigerSHARC processor three-states the $\overline{ADDR31-0}$, $\overline{DATA63-0}$, \overline{MSH}, \overline{MSSD}, $\overline{MS1-0}$, \overline{RD}, \overline{WRL}, \overline{WRH}, \overline{BMS}, \overline{BRST}, \overline{FLYBY}, \overline{TOEN}, \overline{RAS}, \overline{CAS}, \overline{SDWE}, $\overline{SDA10}$, \overline{SDCKE}, \overline{LDQM} and \overline{HDQM} pins and puts the SDRAM into self-refresh mode. The TigerSHARC processor asserts \overline{HBG} until the host deasserts \overline{HBR}. In multiprocessing systems, the current bus master TigerSHARC processor drives \overline{HBG} while all slave DSPs monitor \overline{HBG}. The \overline{HBG} pin has an internal pull-up resistor, which is only enabled on the TigerSHARC processor with ID2-0 = 0. If ID0 is not used, terminate this pin as either pull-up or no connection. If ID7-1 is not used, terminate this pin as pull-up.</p>


Table 5-3. Multiprocessing I/O Pins (Cont'd)

Signal	Description
$\overline{\text{CPA}}$	<p>Core Priority Access</p> <p>Asserted while the TigerSHARC processor's core accesses external memory. This pin allows a slave TigerSHARC processor to interrupt a master TigerSHARC processor's background DMA transfers and gain control of the external bus for core-initiated transactions. $\overline{\text{CPA}}$ is an open drain output, connected to all DSPs in the system. The $\overline{\text{CPA}}$ pin has an internal 500 Ω pull-up resistor, which is only enabled on the TigerSHARC processor with ID2-0 = 0. If ID0 is not used, terminate this pin as either pull-up or no connection. If ID7-1 is not used, terminate this pin as pull-up.</p>
$\overline{\text{DPA}}$	<p>DMA Priority Access</p> <p>Asserted while a high priority TigerSHARC processor DMA channel accesses external memory. This pin allows a high priority DMA channel on a slave TigerSHARC processor, to interrupt transfers of a normal priority DMA channel on a master TigerSHARC processor and gain control of the external bus for DMA-initiated transactions. $\overline{\text{DPA}}$ is an open drain output, connected to all DSPs in the system. The $\overline{\text{DPA}}$ pin has an internal 500Ω pull-up resistor, which is only enabled on the TigerSHARC processor with ID2-0 = 0. If ID0 is not used, terminate this pin as either pull-up or no connection. If ID7-1 is not used, terminate this pin as pull-up.</p>
$\overline{\text{BOFF}}$	<p>Back Off</p> <p>A deadlock situation can occur when the host and a TigerSHARC processor try to read from each other's bus at the same time. When a deadlock occurs, the host can assert $\overline{\text{BOFF}}$ to force the TigerSHARC processor to relinquish the bus before completing the outstanding transaction; but only if the outstanding transaction is to host memory space (MSH).</p>
$\overline{\text{BUSLOCK}}$	<p>Bus Lock Indication</p> <p>Provides an indication that the current bus master has locked the bus.</p>
$\overline{\text{BM}}$	<p>Bus Master</p> <p>The current bus master TigerSHARC processor asserts $\overline{\text{BM}}$. For debugging only. At reset, this is a strap pin. See the <i>ADSP-TS101 TigerSHARC Embedded Processor Data Sheet</i> for more information.</p>
ID2-0	<p>Multiprocessing ID</p> <p>Indicates the TigerSHARC processor's ID. From the ID, the TigerSHARC processor determines its order in a multiprocessor system. These pins also indicate to the TigerSHARC processor which bus request ($\overline{\text{BR7-0}}$) to assert when requesting the bus: 000 = $\overline{\text{BR0}}$, 001 = $\overline{\text{BR1}}$, 010 = $\overline{\text{BR2}}$, 011 = $\overline{\text{BR3}}$, 100 = $\overline{\text{BR4}}$, 101 = $\overline{\text{BR5}}$, 110 = $\overline{\text{BR6}}$, 111 = $\overline{\text{BR7}}$. ID2-0 must have a constant value during system operation and can change during reset only.</p>

Multiprocessing

All on-chip arbiters in a cluster of TigerSHARC processors operate in lock-step on a cycle-by-cycle basis. In this way, every TigerSHARC processor tracks and follows the arbitration sequence of the shared bus. Bus arbitration is accomplished with the use of the $\overline{\text{BR7-0}}$, $\overline{\text{HBR}}$ and $\overline{\text{HBG}}$ pins. The $\overline{\text{BR7-0}}$ pins arbitrate between TigerSHARC processors, and $\overline{\text{HBR}}$ and $\overline{\text{HBG}}$ pins pass control of the bus to the host. If the system has less than eight TigerSHARC processors, any unused $\overline{\text{BRx}}$ pins must be deasserted.

Every TigerSHARC processor has three identification input pins used to distinguish between TigerSHARC processors. These are the ID2-0 pins, where the ID2-0 pins of each TigerSHARC processor are set to a unique number and used as the processor ID. A rotating bus priority is implemented to ensure bus fairness between TigerSHARC processors. After reset, the TigerSHARC processor with ID=000 becomes the master and priority rotates in a round robin fashion, going up from the present master. Priority rotation is interrupted either when the host that has highest priority in the system requests the bus, or when a TigerSHARC processor requests the bus by activating the $\overline{\text{CPA}}$ and $\overline{\text{DPA}}$ (Priority Access) pins.

 In previous SHARC processors (ADSP-2106x and ADSP-2116x families), the use of a processor with ID=001 was mandatory for multiprocessing systems to function correctly. The SHARC processors use ID=001 to select the bus master after reset and reserve ID=000 for single processor systems.

In TigerSHARC processors, the ID pins operate differently. Multiprocessing TigerSHARC systems must always have a processor with ID=000, which is the bus master after reset. Under some circumstances, multiprocessing TigerSHARC systems cannot function properly unless a processor with ID=000 is present. For example, the ID=000 processor must be present in any system including SDRAM because this processor performs the initialization (Mode Register Set MRS) of the SDRAM. Also, there are issues related to open drain pull-ups which are only enabled on the processor with ID=000.

The TigerSHARC processor can lock the bus in certain cases. For more information, see “Bus Lock” on page 5-47. The following arbitration flow occurs when there is no bus lock.

System priority in ascending order is:

- Host
- TigerSHARC processor ($\overline{\text{BR}}$ along with $\overline{\text{CPA}}$ asserted)
- TigerSHARC processor ($\overline{\text{BR}}$ along with $\overline{\text{DPA}}$ asserted)
- TigerSHARC processor ($\overline{\text{BR}}$ asserted)

The current master yields the bus under these conditions:

- The current master does not request the bus and another slave asserts its $\overline{\text{BR}}$ or $\overline{\text{HBR}}$.
- The current master requests the bus, but another slave asserts its $\overline{\text{BR}}$ and $\overline{\text{CPA}}$. The current master does not assert $\overline{\text{CPA}}$.
- The current master requests the bus, but another slave asserts its $\overline{\text{BR}}$ and $\overline{\text{DPA}}$. The current master does not assert $\overline{\text{CPA}}$ or $\overline{\text{DPA}}$.
- The current master requests the bus, but the host asserted its $\overline{\text{HBR}}$.
- The current master requests the bus, but the value of the BMAX register expires and another slave asserts its $\overline{\text{BR}}$. The BMAX register holds a count of cycles for which the TigerSHARC processor may keep the bus. The count is in internal cycles (CCLK frequency). See “BMAX Register” on page 2-39 for more details.

Bus Arbitration Protocol

When one of the TigerSHARC processors needs to become a master, it asserts its own $\overline{\text{BR}}$ line, while at the same time monitoring the other $\overline{\text{BR}}$ lines. The current master, after completing its transaction, deasserts its

Multiprocessing

own \overline{BR} . Then, bus mastership is passed to the new requester. The new bus master retains the bus mastership by maintaining its \overline{BR} , which is asserted continuously. When the current master relinquishes the bus, it is assigned the lowest priority. When other slaves want to become masters, each of them asserts their own \overline{BR} and the slave with the highest priority gains the mastership of the bus. The remaining competing slaves keep their \overline{BR} s asserted until gaining bus mastership according to their priority. There is no pipeline between masters. The current master completes all its pending transactions (transactions that have begun executing on the cluster bus) before relinquishing the bus. Because of master change cycles, an IDLE cycle occurs when mastership is passed from the current master to a new master.

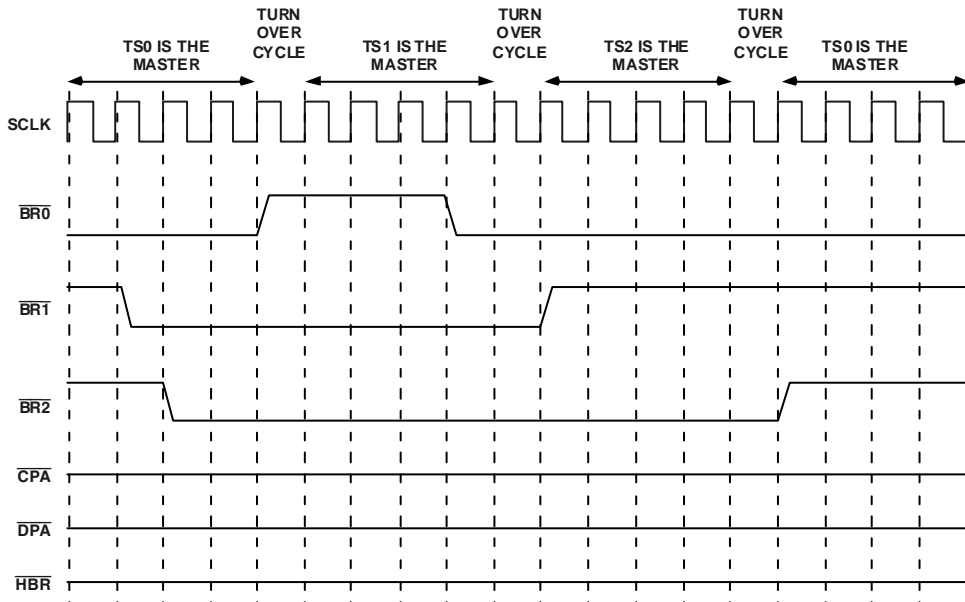


Figure 5-20. External Bus Arbitration Sequence, $\overline{CPA}/\overline{DPA}/\overline{HBR}$ Inactive

The host can be a bus master on the TigerSHARC processor's external bus—it uses $\overline{\text{HBR}}$ and $\overline{\text{HBG}}$ for gaining control of the bus. In order for the host to become bus master, the host must assert $\overline{\text{HBR}}$ and wait until $\overline{\text{HBG}}$ is asserted by the current TigerSHARC processor master. The TigerSHARC processor relinquishes the external bus and indicates this by asserting $\overline{\text{HBG}}$.

After completing all outstanding transactions (unless the backoff mechanism is asserted—see “Backoff” on page 5-50), the host keeps $\overline{\text{HBR}}$ asserted for as long as it needs the bus. The master that last relinquished the bus keeps its $\overline{\text{BR}}$ line asserted, so that when the host deasserts $\overline{\text{HBR}}$, it becomes the master again.

The bus arbitration allows any TigerSHARC processor to gain temporary priority over the current master. This is achieved by two wired OR pins— $\overline{\text{CPA}}$ and $\overline{\text{DPA}}$.

Core Priority Access (CPA)

The Core Priority Access pin, $\overline{\text{CPA}}$, is asserted when the TigerSHARC processor core accesses external memory. This allows a slave TigerSHARC processor to interrupt background transfers of a DMA channel belonging to a master TigerSHARC processor and gain control of the external bus. The current master in this case terminates its transaction and passes the bus mastership to the requesting TigerSHARC processor by deasserting its $\overline{\text{BR}}$. When $\overline{\text{CPA}}$ is asserted, only TigerSHARC processors with core transactions can request the bus. The other requesting TigerSHARC processors deassert their $\overline{\text{BR}}$ s when they sense that $\overline{\text{CPA}}$ is asserted. When more than one TigerSHARC processor requests the bus by asserting their $\overline{\text{BR}}$ s along with $\overline{\text{CPA}}$, the TigerSHARC processor with the highest priority gains bus mastership.

DMA Priority Access (DPA)

The DMA Priority Access, $\overline{\text{DPA}}$, is asserted when a TigerSHARC processor high priority DMA channel accesses external memory—only if $\overline{\text{CPA}}$ is not asserted. This allows a high priority DMA channel belonging to a slave

Multiprocessing

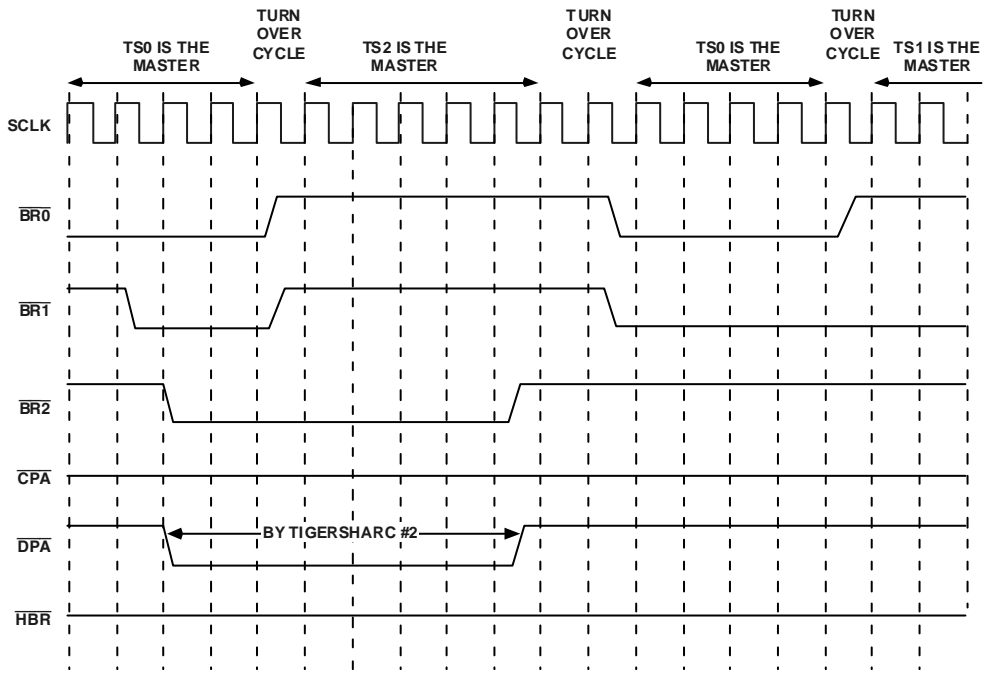


Figure 5-21. External Bus Arbitration Sequence; Only One Active \overline{DPA} ; \overline{HBR} Inactive

TigerSHARC processor to interrupt background transfers of a regular priority DMA channel belonging to a master TigerSHARC processor and gain control of the external bus. The current master in this case terminates its transaction and passes the bus mastership to the requesting TigerSHARC processor by deasserting its \overline{BR} . When \overline{DPA} is asserted, only TigerSHARC processors with high priority DMA transactions can request the bus. The other requesting TigerSHARC processors deassert their \overline{BR} s when they sense \overline{DPA} is asserted. When more than one TigerSHARC processor requests the bus by asserting their \overline{BR} s along with \overline{DPA} , the TigerSHARC processor with the highest priority gains the bus mastership.

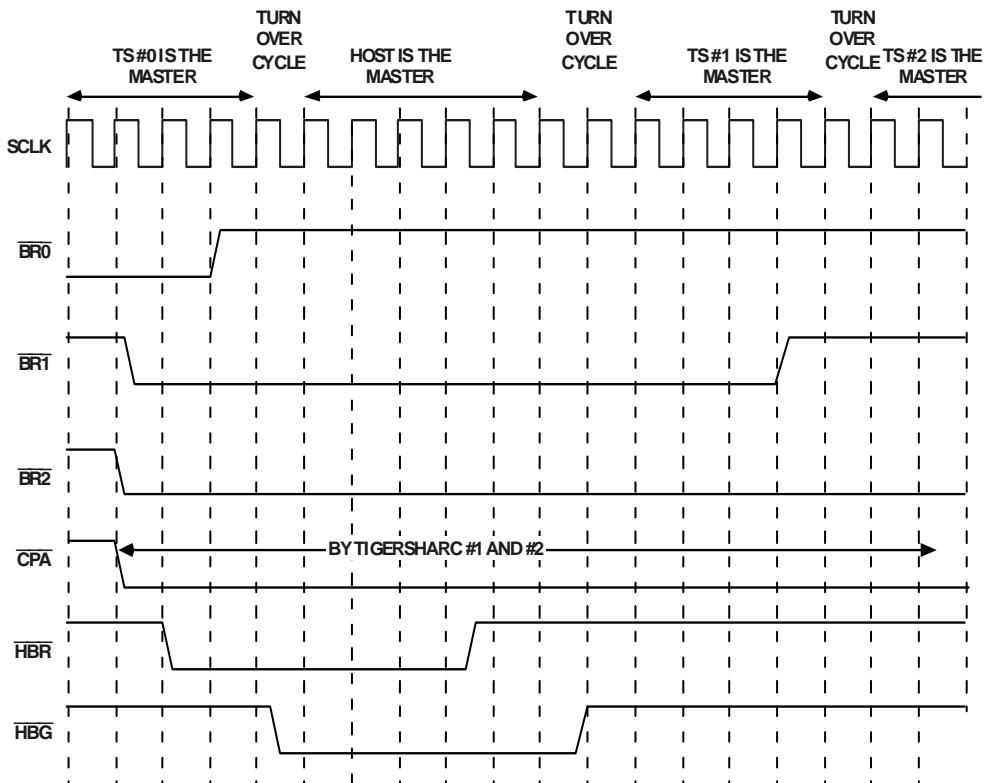



Figure 5-22. External Bus Arbitration Sequence; More Than One Active \overline{CPA} ; \overline{HBR} Active

When a priority access (\overline{CPA} or \overline{DPA}) finishes, and no other priority access is being requested – there are two idle cycles for the previous master (that was interrupted) to gain control of the bus. One cycle is to sense that the priority access is over, the second is to take control of the bus. This is illustrated in Figure 5-22 on page 5-45.


Multiprocessing

The host can begin driving the bus in the same cycle or in any cycle after $\overline{\text{HBG}}$ is sampled asserted. If the host does not drive the bus immediately after $\overline{\text{HBG}}$ is sampled asserted, TigerSHARC pull-ups will stabilize bus signals in the interim. All signals are sampled on the rising edge of SCLK.

The host must stop driving the bus no later than the same cycle that $\overline{\text{HBR}}$ is deasserted. The host can stop driving the bus in any of the cycles preceding $\overline{\text{HBR}}$ deassertion as well. The host should deassert $\overline{\text{HBR}}$ only after the last transaction has completed (RD/WR strobes should be deasserted with $\overline{\text{HBR}}$ - TigerSHARC pull-ups will stabilize bus signals at this point until the new bus master drives them).

-  In the arbitration process, requesting bus agents with an active $\overline{\text{CPA}}$ have higher priority over requesting bus agents with an active $\overline{\text{DPA}}$. This implies that requesting bus agents with an active $\overline{\text{DPA}}$ deassert their $\overline{\text{BR}}$ s upon sensing that a $\overline{\text{CPA}}$ is asserted by other bus agents.

Once the bus is owned by the TigerSHARC processor with the bus lock, the bus lock priority is almost the highest in the system. The TigerSHARC processor relinquishes the bus when the $\overline{\text{HBR}}$ and $\overline{\text{BOFF}}$ pins are asserted according to the protocol described in “Backoff” on page 5-50. When bus lock is set, the bus is not relinquished by either a $\overline{\text{CPA}}$, $\overline{\text{DPA}}$, or a $\overline{\text{HBR}}$ assertion, nor by the BMAX count expire.

-  In an TigerSHARC processor cluster system, when one of the slave DSPs performs a priority access $\overline{\text{CPA}}$ or $\overline{\text{DPA}}$, the TigerSHARC processor that has control of the bus deasserts the bus request line. Once the priority access is completed, there are two turnover cycles for the original TigerSHARC processor to regain control of the bus—one for the TigerSHARC processor to sense the priority access is done, and another to take control of the bus. This is illustrated in Figure 5-22 on page 5-45, which shows only one turnover cycle. This is because there are two priority accesses being performed. It only takes one turnover cycle to transfer from one priority to another.

Bus Fairness — BMAX

The system designer may want to limit the time a single TigerSHARC processor holds the bus, in order to prevent bus starvation of other masters in the system. The `BMAX` register defines the period that the TigerSHARC processor may own the bus. The time is defined in `CCLK` cycles. The `BMAX` counter is initiated to the defined value when the TigerSHARC processor gets hold of the bus and starts counting down. Countdown is paused while the host gets $\overline{\text{HBG}}$, so the amount of time that the host is the bus master is not included as a part of the specific TigerSHARC processor share in the bus. When the `BMAX` count expires and there is another bus request asserted by another TigerSHARC processor in the system, the master TigerSHARC processor relinquishes the bus for at least one cycle. If no other bus request is asserted, the `BMAX` counter is reloaded and the master continues to own the bus.

The `BMAX` count is not precise. Due to synchronization between clock domains, there may be a skew of up to three `LCLK` cycles plus three `SCLK` cycles in the period. Additionally, when the `BMAX` count expires, the bus is not relinquished until the current transaction has been completed. When the bus lock (`BUSLK`) bit in the `BUSLK` register is set, the `BMAX` expire is ignored until the bus lock is cleared. Once it is cleared, the bus is relinquished.

Bus Lock

The bus lock feature is implemented in order to support atomic read-modify-write operations. When a TigerSHARC processor needs to access a semaphore, it requests a bus lock by setting its `BUSLK` bit in the `BUSLK` register (see “`BUSLK` System Control” on page 2-38). Setting this bit causes the bus arbitration logic to request the bus; and maintains $\overline{\text{BR}}$ assertion as long as the `BUSLK` bit is set. Once the `BUSLK` bit is cleared, arbitration resumes the regular path. Before accessing a semaphore, the TigerSHARC processor should check whether it is the bus master by read-

Multiprocessing

ing the `CURRENT_BUS_MASTER` field bit in the `SYSTAT` register. If the TigerSHARC processor is the current master, writing to the semaphore location can be executed safely.

The system indicates whether or not the bus is locked by the TigerSHARC processor via a `BUSLOCK` pin. This enables the system to propagate the bus lock indication by way of a bridge to another bus.

There is also support for bus lock in the interrupt mechanism. When the bus lock bit is set and the TigerSHARC processor gets hold of the bus, a bus lock interrupt is set. For more information, see “Bus Lock Interrupt” on page 4-7.

Host Interface

The host interface connects a host to the TigerSHARC processor’s external bus. When the host is a slave, it uses the same protocol as the TigerSHARC processor—pipelined or slow device protocol depending on the host interface configuration specified in the `SYSCON` register. The host can access the TigerSHARC processor in the same pipelined protocol as any other TigerSHARC processor. The host can also access any of the slaves in the system using its own protocol. There are some slight differences between the host’s behavior and the TigerSHARC processor’s behavior. These differences are detailed below.

When a transaction to any address space (except the host space) follows a read from the host, the non-host transaction is serialized—for example, the non-host transaction takes place only after all host transactions have completely terminated. This is illustrated in Figure 5-23.

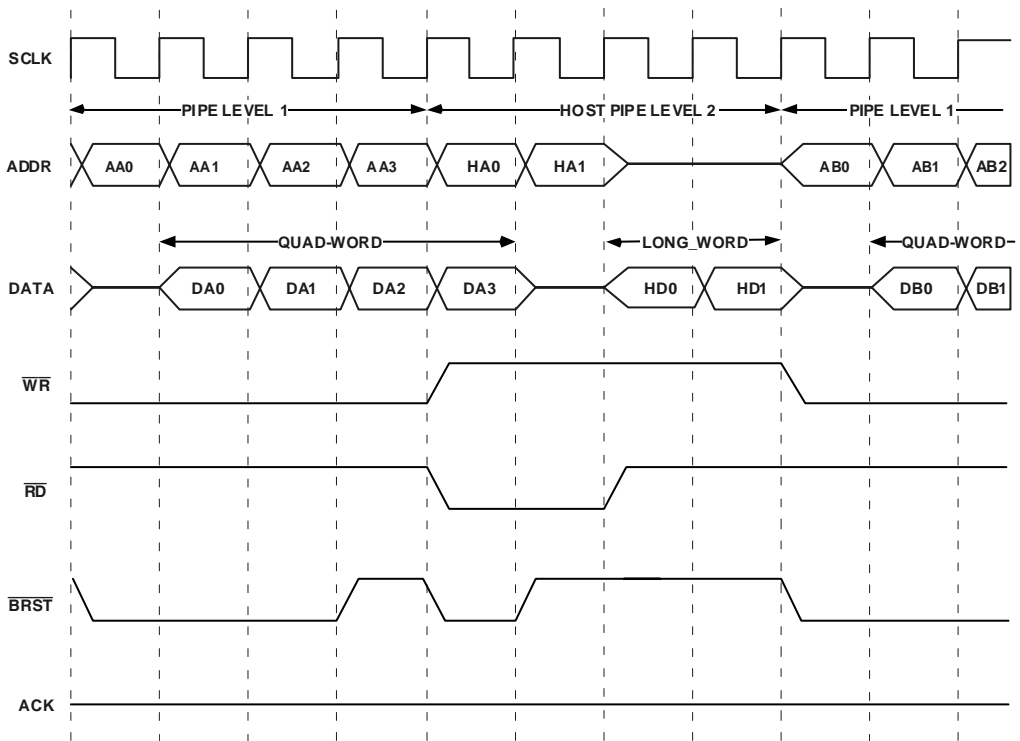


Figure 5-23. Write to Host Transaction Followed by a Read Cycle From the Host by the TigerSHARC Processor, Bus Width = 32

When the host accesses a TigerSHARC processor, it should comply with all the rules of TigerSHARC processor pipelined transactions. The most important rules are:

- Pipeline depth is always four for read and one for write.
- There are IDLE cycle restrictions.
- No read from broadcast address space is allowed.
- The host must operate with an identical system configuration as other TigerSHARC processors on the cluster bus.

Multiprocessing


- If the host uses SDRAM, the TigerSHARC processor should not use SDRAM before the TigerSHARC processor (ID #0) has initialized the SDRAM. (see “SDRAM Programming” on page 6-19).
- If the host uses SDRAM, it puts the SDRAM in self-refresh mode before it relinquishes the bus to the host. The host, in order to use the SDRAM, should change the SDRAM mode to normal work. Before returning the bus to the TigerSHARC processor, the host should put the SDRAM into self-refresh mode again.

When the host executes burst transactions to and from the TigerSHARC processor, the TigerSHARC processor allows for a continuous burst of more than four data units. The host issues a starting burst address. As long as $\overline{\text{BRST}}$ is asserted, the TigerSHARC processor increments the address internally. The $\overline{\text{BRST}}$ pin is used to indicate transaction continuation—in this case the transaction is ended on every quad-word address. The first transaction should begin and the last transaction should end in quad-word aligned addresses.

Backoff

The host, one of the masters in the system, can request the bus, as explained in “Bus Arbitration Protocol” on page 5-41. In some cases, a deadlock situation may occur. This happens when the host and the TigerSHARC processor are trying to read from each other’s bus at the same time—see Figure 5-24 on page 5-52. In this case, the TigerSHARC processor has control of its system bus, the host has control of its system bus, and both issue a read transaction from each other. This results in the host bridge requesting both buses to execute both transactions. Neither of the buses is relinquished before the current master completes its own transaction. Likewise, neither of the transactions is completed because the buses are not relinquished. To resolve the deadlock, one of the requesters has to give up the bus, allowing the other requester to complete its read transaction. The TigerSHARC processor is designed to yield the bus to the host when the latter asserts $\overline{\text{BOFF}}$, signaling the TigerSHARC processor to yield.

In response, the TigerSHARC processor asserts \overline{HBG} , allowing the host to execute its transactions. After the host completes the transactions and relinquishes the bus, the TigerSHARC processor performs the read transactions that were suspended when \overline{BOFF} was asserted.

 The backoff mechanism may not be used to stop write transactions to host.

The backoff mechanism can only be active when the TigerSHARC processor has reached the data cycle of transactions targeted to the host; this occurs after the host has deasserted the ACK pin. If the host is programmed to be accessed by the TigerSHARC processor in slow protocol with zero wait states, the backoff mechanism cannot be used because in this case the ACK signal cannot be asserted for the transaction.

Backoff also overrides a bus lock. If the host can access resources that are protected by a semaphore, there should be a system solution for the semaphore. One possible solution is to implement semaphores on the bridge in a way that does not require bus lock. If the semaphore is set when read and the read data is the previous value, there is no need for bus lock on the semaphore access. The host interface can be notified of a bus lock by the $\overline{BUSLOCK}$ pin.

Multiprocessing

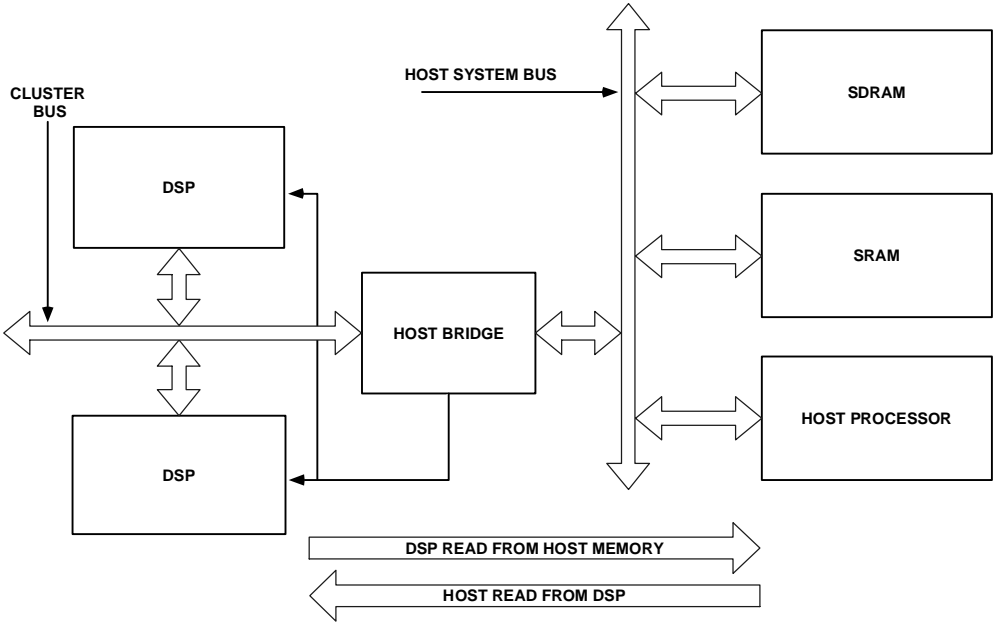


Figure 5-24. Deadlock Scenario

6 SDRAM INTERFACE

The TigerSHARC processor has a dedicated address space to support SDRAM. This address space is selected by the $\overline{\text{MSSD}}$ (Memory Select SDRAM), which can be used for direct interface with SDRAM devices. The SDRAM is accessed on memory address range 0x04000000 to 0x07FFFFFF (see “External Memory Bank Space” on page 2-4).

SDRAM, unlike conventional DRAM, is synchronous to the SCLK (system clock). All inputs are sampled and all outputs are valid at the rising edge of the clock. The synchronous interface allows data transfer every cycle, yielding high throughput up to 400M Bytes/second for a 32-bit bus width, and 800M Bytes/second for a 64-bit bus width. The SDRAM has several types of burst accesses, depending on the initialization of its mode register, SDRCON . Each access type is preceded by issuance of the appropriate command to the SDRAM. The SDRAM specific modes should be initialized in the SDRCON register. (See “SDRCON (SDRAM Configuration) (DMA 0x180484)” on page 2-36.) The SDRAM interface provides a glueless interface with standard SDRAMs—6M bits, 64M bits, 128M bits, 256M bits, and 512M bits. The TigerSHARC processor directly supports a maximum of 64M words x 32 bits of SDRAM.

For SDRAM, all initialization MRS (Mode Register Set) command and configuration is done by processor with ID000. This ID should always be present in any TigerSHARC processor cluster.

The SDRAM is organized internally into two or four banks. The SDRAM Bank Select pins determine which bank is being addressed. The SDRAM has a programmable read latency parameter that must be initialized by the application according to the type of device and the operating clock frequency.

In order to meet the demanding SDRAM timing requirements, the TigerSHARC processor allows the SDRAM address and controls to be pipelined. The pipeline depth bit in the `SDRCON` register is used for this purpose. When this bit is set, data in write accesses are delayed by one cycle, allowing the address and controls to be externally latched and optionally manipulated in one cycle (for example, selection between Dual In-line Memory Modules (DIMMs) in the array). In pipelined read accesses, data is sampled by the TigerSHARC processor one cycle later. ($\overline{\text{CAS}}$ latency plus pipeline cycle are maximum four cycles.) (Refer to the timing diagram in the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*.) Pipelining can be used when connecting several SDRAM devices in parallel such that their collective load is too high to be driven by the TigerSHARC processor.

SDRAM chips are available from different vendors. Each vendor has variations on timing requirements pertaining to `ACT` to `PRE` command delays (t_{RAS}) and `PRE` to `ACT` command delays (t_{RP}). In order to support all major vendors and different speed grades, the `SDRCON` register is programmed to help the system designer meet the specific SDRAM timing requirements. The bit fields are $\overline{\text{CAS}}$ latency, `PRE` to $\overline{\text{RAS}}$ delay, and $\overline{\text{RAS}}$ to `PRE` delay. See “`SDRCON` (SDRAM Configuration) (DMA 0x180484)” on page 2-36 for more information regarding `SDRCON` register definitions.

The TigerSHARC processor programs the SDRAM to full-page bursts. This allows other bus agents that have their own SDRAM controller, for example, host, to perform full-page burst transactions. The TigerSHARC processor itself operates in bursts of up to four data units. Whenever possible, however, it issues a new $\overline{\text{CAS}}$ for the following transaction early enough to make the whole sequence look like a continuous burst.

The TigerSHARC processor includes a programmable refresh rate in order to meet refresh timing. This rate may be used to coordinate between the clock rate and the required refresh rate. See “Setting the Refresh Counter Value (Refresh Rate)” on page 6-25.

The following are definitions used throughout this chapter.

- **Bank Activate Command.** Activates the selected bank and latches in a new row address. It must be applied before a read or write command.
- **Burst Length.** Determines the number of words that the SDRAM inputs or outputs after detecting a write or read command, respectively. Although the SDRAM device is programmed for full-page burst, the controller uses quad-word (128 bits) burst mode. For 32-bit bus width, the burst length is 4 words, and for 64-bit width, the burst length is 2 words. Only the first read or write command is accompanied with an external address that is driven by the controller until the burst is interrupted by another address.
- **Burst Type.** Determines the order in which the SDRAM delivers or stores burst data after detecting a read or write command, respectively. The processor supports sequential accesses only.
- **$\overline{\text{CAS}}$ Latency.** The delay, in clock cycles, between when the SDRAM detects the read command and when it provides the data at its output pins. The speed grade of the device and the application's clock frequency determine the value of the $\overline{\text{CAS}}$ latency.
- **The application must program the $\overline{\text{CAS}}$ latency value into the SDRCON register.**
- **CBR Automatic Refresh ($\overline{\text{CAS}}$ before $\overline{\text{RAS}}$) Mode.** In this mode, the SDRAM drives its own refresh cycle with no external control input. At cycle end, all SDRAM banks are precharged (IDLE).

- **HDQM/LDQM Data I/O Mask Function.** The HDQM/LDQM pins are used by the controller to mask write operations. See “Understanding DQM Operation” on page 6-29.
- **SDRCON Register.** An IOP register that contains programmable SDRAM control and configuration parameters that support different vendor's timing and power up sequence requirements.
- **Mode Register.** The SDRAM configuration register that contains user-defined parameters corresponding to the processor SDRCON register. After initial power up and before executing a read or write command, the application must program the initialization sequence in the SDRCON register.
- **Page Size.** The size, in words, of the SDRAM page. The processor supports 1024-, 512-, and 256-word page sizes. Page size is a programmable option in the SDRCON register.
- **Precharge Command.** Precharges an active bank.
- **Refresh Rate.** Programmable value in the SDRCON register. The clock supplied to the SDRAM can vary between 50 and 100 MHz. The refresh rate enables applications to coordinate SCLK rate with the SDRAM's required refresh rate.
- **Self-Refresh.** The SDRAM internal timer initiates automatic refresh cycles periodically, without external control input. This command places the SDRAM device in a low power mode.
- **t_{RAS} .** Active Command time. Required delay between issuing an activate command and issuing a precharge command. A vendor-specific value. This option is programmable in the SDRCON register.

- t_{RC} . Bank Cycle time. The required delay between successive Bank Activate commands to the same bank. This vendor-specific value is defined as $t_{RC} = t_{RP} + t_{RAS}$. The processor fixes the value of this parameter so it is a non-programmable option.
- t_{RCD} . \overline{RAS} to \overline{CAS} Delay. The required delay between an ACT command and the start of the first read or write operation. This vendor-specific value is defined as $t_{RCD} = CL$. The processor fixes the value of this parameter so it is a non-programmable option.
- t_{RP} . Precharge Time. The required delay between issuing a precharge command and issuing an activate command. This vendor-specific value is programmable in `SDRCON`.

Figure 6-1 on page 6-6 shows the SDRAM controller interface between the internal TigerSHARC processor core and the external SDRAM device.

The TigerSHARC processor normally generates an external memory address, which then asserts the SDRAM memory select line (\overline{MSSD}), along with \overline{RD} and \overline{WR} accesses. These control signals are intercepted by the SDRAM controller. The memory access to SDRAM is based on the mapping of the addresses and memory selects. The configuration is programmed in the `SDRCON` register. The SDRAM controller can hold off the TigerSHARC processor core or I/O processor as determined by refresh, non-sequential access, or page miss latency overhead.

The SDRAM controller provides a glueless interconnection between the SDRAM control, address, and data pins and the TigerSHARC processor's internal Harvard Architecture buses. The internal 32-bit address bus is multiplexed by the SDRAM controller to generate the corresponding chip select, row address, column address, and bank select signals to the SDRAM.

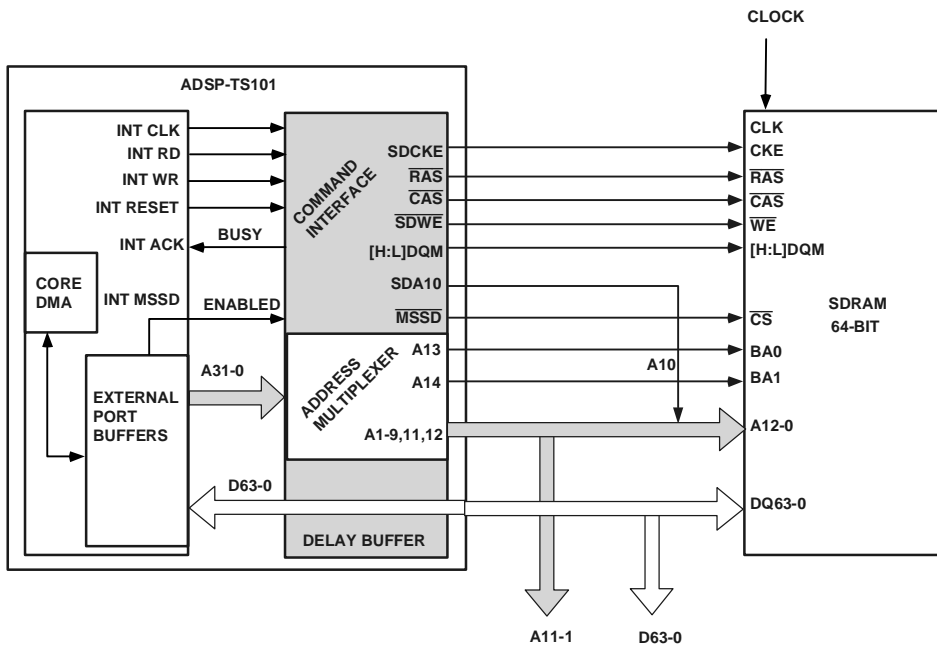


Figure 6-1. SDRAM Controller Interface
(for a 64-bit Bus Configuration)

SDRAM I/O Pins

Table 6-1 lists the I/O pins associated with the SDRAM. Other bus interface I/O pins are detailed in Table 5-1 on page 5-4.

Table 6-1. SDRAM I/O Pins

Signal	Description
ADDR31-0	<p>Address Bus</p> <p>The TigerSHARC processor issues addresses for accessing memory and peripherals on these pins. In a multiprocessor system, the bus master drives addresses for accessing internal memory or I/O processor registers of other ADSP-TS101S DSPs. The TigerSHARC processor inputs addresses when a host or another TigerSHARC processor accesses its internal memory or I/O processor registers.</p>
DATA63-0	<p>External Data Bus</p> <p>Data and instructions are received and driven by the TigerSHARC processor, on these pins.</p>
\overline{MSSD}	<p>Memory Select SDRAM</p> <p>\overline{MSSD} is asserted whenever the TigerSHARC processor accesses SDRAM memory space. \overline{MSSD} is a decoded memory address pin that is asserted whenever the TigerSHARC processor issues an SDRAM command cycle access to ADDR31-26 (= 0b000001). \overline{MSSD} in a multiprocessor system is driven by the master TigerSHARC processor.</p>
\overline{RAS}	<p>Row Address Strobe</p> <p>When sampled low, \overline{RAS} indicates that a row address is valid in a read or write of SDRAM. In other SDRAM accesses, \overline{RAS} defines the type of operation to execute according to SDRAM specification.</p>
\overline{CAS}	<p>Column Address Strobe</p> <p>When sampled low, \overline{CAS} indicates that a column address is valid in a read or write of SDRAM. In other SDRAM accesses, \overline{CAS} defines the type of operation to execute according to the SDRAM specification.</p>
LDQM	<p>Low Word SDRAM Data Mask</p> <p>When LDQM is sampled high, the TigerSHARC processor three-states the SDRAM DQ buffers. LDQM is valid on SDRAM transactions when \overline{CAS} is asserted and is inactive on read transactions. On write transactions, LDQM is active when accessing an odd address word on a 64-bit memory bus to disable the write of the low word.</p>

SDRAM Physical Connection

Table 6-1. SDRAM I/O Pins (Cont'd)

Signal	Description
HDQM	High Word SDRAM Data Mask When HDQM is sampled high, the TigerSHARC processor three-states the SDRAM DQ buffers. HDQM is valid on SDRAM transactions when $\overline{\text{CAS}}$ is asserted and is inactive on read transactions. On write transactions, HDQM is active when accessing an even address in word accesses or is active when memory is configured for a 32-bit bus to disable the write of the high word.
$\overline{\text{SDWE}}$	SDRAM Write Enable When sampled low while $\overline{\text{CAS}}$ is active, $\overline{\text{SDWE}}$ indicates an SDRAM write access. When sampled high while $\overline{\text{CAS}}$ is active, $\overline{\text{SDWE}}$ indicates an SDRAM read access. In other SDRAM accesses, $\overline{\text{SDWE}}$ defines the type of operation to execute according to SDRAM specification.
SDCKE	SDRAM Clock Enable This pin activates the SDRAM clock for SDRAM self-refresh or suspend modes. A slave TigerSHARC processor in a multiprocessor system does not have the pull-up or pull-down. A master TigerSHARC processor (or ID = 0 in a single processor system) has a 100 k Ω pull-up before granting the bus to the host, except when the SDRAM is put in self-refresh mode. In self-refresh mode, the master has a 100 k Ω pull-down before granting the bus to the host.
SDA10	SDRAM Address Bit10 Separate A10 signals enable SDRAM refresh operation while the TigerSHARC processor executes non-SDRAM transactions.

SDRAM Physical Connection

The SDRAM address and data pins can be connected directly to the TigerSHARC processor address and data pins. The data bus is either the full 64 bits or the bottom 32 bits according to the memory bus width programming in SYSCON (see “SYSCON Programming” on page 5-11). The

address connection is according to the bus width and there is an option to split the load on different pins. For 32-bit bus, the connection is listed below.

- SDRAM address Bits9–0 are connected to TigerSHARC processor ADDR9–0
- SDRAM address Bit10 is connected to TigerSHARC processor pin SDA10
- SDRAM address Bits15–11 are connected to TigerSHARC processor ADDR15–11 (or as many as required)

In a 64-bit bus, the address should be shifted. The scheme should be as follows.

- SDRAM address Bits9–0 are connected to TigerSHARC processor ADDR10–1
- SDRAM address Bit10 is connected to TigerSHARC processor pin SDA10
- SDRAM address Bits14–11 are connected to TigerSHARC processor ADDR15–12 (or as many as required)

The control signals are specific SDRAM control signals. These signals are listed below.

- $\overline{\text{MSSD}}$ – SDRAM Chip Select
- $\overline{\text{RAS}}$ – Row Address Strobe
- $\overline{\text{CAS}}$ – Column Address Strobe
- $\overline{\text{SDWE}}$ – SDRAM Write Enable
- LDQM – Data Mask for Low Word (data Bits31–0)

SDRAM Physical Connection

- HDQM – Data Mask for High Word (data Bits63–2)
- SDCKE – Clock Enable

In large memory systems, there are often many SDRAM chips that can overload the address bus. The scheme for a 32-bit connection is as follows.

- SDRAM address Bits9–0 are connected to TigerSHARC processor ADDR25–16
- SDRAM address Bits15–0 are duplicated on TigerSHARC processor ADDR31–16
- SDRAM address Bit10 is connected to TigerSHARC processor pin SDA10
- SDRAM address Bits15–11 are connected to TigerSHARC processor ADDR31–27 (or as many as required)

The scheme for a 64-bit bus is as follows.

- SDRAM address Bits9–0 are connected to TigerSHARC processor ADDR26–17
- SDRAM address Bit10 is connected to TigerSHARC processor pin SDA10
- SDRAM address Bits14–11 are connected to TigerSHARC processor ADDR31–28 (or as many as required)

Another option to reduce the load is to buffer the address and control signals in registers to delay these signals in a cycle. In this case the pipeline depth bit in SDRCON should be set.

Internal TigerSHARC processor Address and SDRAM Physical Connection

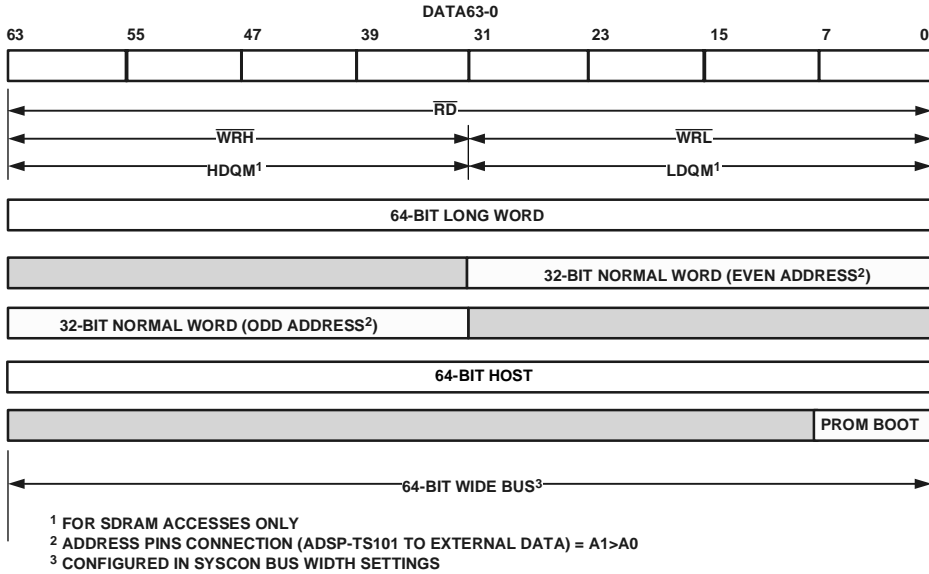


Figure 6-2. External Port Data Align for 64-bit

SDRAM Physical Connection

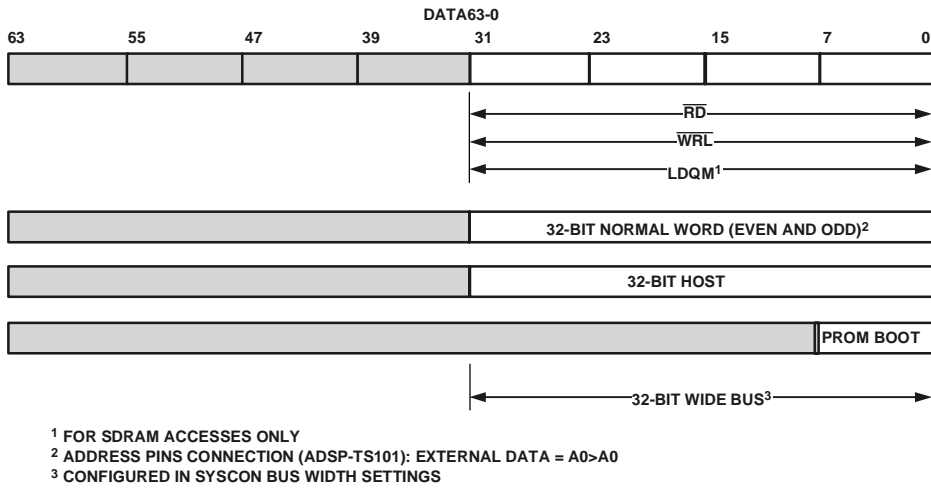


Figure 6-3. External Port Data Align for 32-bit

Table 6-2. Word Page Size 256, 64-Bit Bus Width

Physical TigerSHARC Processor Pin	Bank Active Cycle Internal Address	Column Access Cycle Internal Address	Physical SDRAM Pin
A0	8	0	NC
A1	9	1	A0
A2	10	2	A1
A3	11	3	A2
A4	12	4	A3
A5	13	5	A4
A6	14	6	A5
A7	15	7	A6
A8	16	8	A7
A9	17	9	A8
A10	18	10	A9
SDA10	19	Zero	A10/AP
A11	Irrelevant	Irrelevant	NC
A12	20	20	A11 or Bank
A13	21	21	A12 or Bank
A14	22	22	A13 or Bank
A15	23	23	A14 or Bank

SDRAM Physical Connection

Table 6-3. Word Page Size 256, 32-Bit Bus Width

Physical TigerSHARC Processor Pin	Bank Active Cycle Internal Address	Column Access Cycle Internal Address	Physical SDRAM Pin
A0	8	0	A0
A1	9	1	A1
A2	10	2	A2
A3	11	3	A3
A4	12	4	A4
A5	13	5	A5
A6	14	6	A6
A7	15	7	A7
A8	16	8	A8
A9	17	9	A9
A10	Irrelevant	Irrelevant	NC
SDA10	18	0	A10/AP
A11	19	19	A11 or Bank
A12	20	20	A12 or Bank
A13	21	21	A13 or Bank
A14	22	22	A14 or Bank
A15	23	23	A15 or Bank

Table 6-4. Word Page Size 512, 32-Bit Bus Width

Physical TigerSHARC Processor Pin	Bank Active Cycle Internal Address	Column Access Cycle Internal Address	Physical SDRAM Pin
A0	9	0	A0
A1	10	1	A1
A2	11	2	A2
A3	12	3	A3
A4	13	4	A4
A5	14	5	A5
A6	15	6	A6
A7	16	7	A7
A8	17	8	A8
A9	18	9	A9
A10	Irrelevant	Irrelevant	NC
SDA10	19	Zero	A10/AP
A11	20	20	A11 or Bank
A12	21	21	A12 or Bank
A13	22	22	A13 or Bank
A14	23	23	A14 or Bank
A15	24	24	A15 or Bank

SDRAM Physical Connection

Table 6-5. Word Page Size 512, 64-Bit Bus Width

Physical TigerSHARC Processor Pin	Bank Active Cycle Internal Address	Column Access Cycle Internal Address	Physical SDRAM Pin
A0	9	0	NC
A1	10	1	A0
A2	11	2	A1
A3	12	3	A2
A4	13	4	A3
A5	14	5	A4
A6	15	6	A5
A7	16	7	A6
A8	17	8	A7
A9	18	9	A8
A10	19	10	A9
SDA10	20	Zero	A10/AP
A11	Irrelevant	Irrelevant	NC
A12	21	21	A11 or Bank
A13	22	22	A12 or Bank
A14	23	23	A13 or Bank
A15	24	24	A14 or Bank

Table 6-6. Word Page Size 1k, 32-Bit Bus Width

Physical TigerSHARC Processor Pin	Bank Active Cycle Internal Address	Column Access Cycle Internal Address	Physical SDRAM Pin
A0	10	0	A0
A1	11	1	A1
A2	12	2	A2
A3	13	3	A3
A4	14	4	A4
A5	15	5	A5
A6	16	6	A6
A7	17	7	A7
A8	18	8	A8
A9	19	9	A9
A10	Irrelevant	Irrelevant	NC
SDA10	20	Zero	A10/AP
A11	21	21	A11 or Bank
A12	22	22	A12 or Bank
A13	23	23	A13 or Bank
A14	24	24	A14 or Bank
A15	25	25	A15 or Bank

SDRAM Physical Connection

Table 6-7. Word Page Size 1k, 64-Bit Bus Width

Physical TigerSHARC Processor Pin	Bank Active Cycle Internal Address	Column Access Cycle Internal Address	Physical SDRAM Pin
A0	10	0	NC
A1	11	1	A0
A2	12	2	A1
A3	13	3	A2
A4	14	4	A3
A5	15	5	A4
A6	16	6	A5
A7	17	7	A6
A8	18	8	A7
A9	19	9	A8
A10	20	10	A9
SDA10	21	Zero	A10/AP
A11	Irrelevant	Irrelevant	NC
A12	22	22	A11 or Bank
A13	23	23	A12 or Bank
A14	24	24	A13 or Bank
A15	25	One	A14 or Bank

SDRAM Programming

SDRAM Control Register (SDRCON)

SDRAMs are available from several vendors. Each vendor has different SDRAM product requirements for the power up sequence and the timing parameters— t_{RAS} (ACT to PRE command delay), t_{RCD} and t_{RP} (PRE to ACT command delay). Use only SDRAMs that comply with Joint Electronic Device Engineering Council (JEDEC) specifications. In order to support multiple vendors, the TigerSHARC processor SDRCON register can be programmed to meet these requirements.

Figure 6-4 on page 6-20 and Figure 6-5 on page 6-21 show the SDRAM control register.

SDRAM Programming

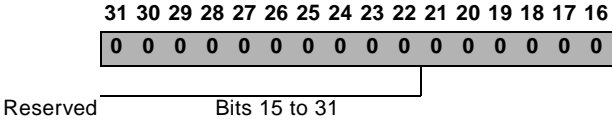


Figure 6-4. SDRCON (Upper) Register Bit Descriptions

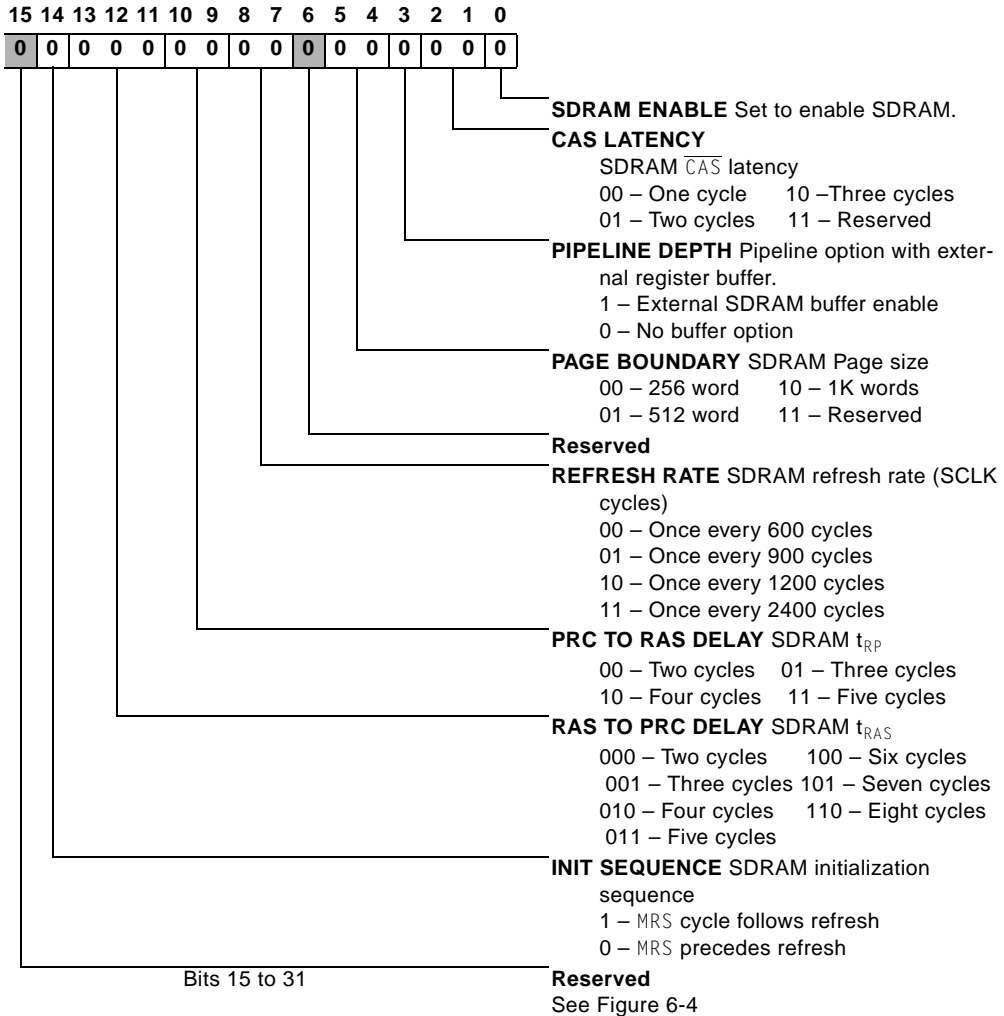


Figure 6-5. SDRCON (Lower) Register Bit Descriptions

The SDRCON register of the TigerSHARC processor stores the configuration information of the SDRAM interface. Writing configuration parameters initiates commands to the SDRAM that take effect immediately.

SDRAM Programming

In the `SDRCON` register, set the parameter bits as follows.

- Enable the SDRAM.
- Select the $\overline{\text{CAS}}$ latency value (CL)
- Set the SDRAM buffering option (pipeline depth)
- Select the SDRAM page size (page boundary)
- Select the refresh counter value (refresh rate)
- Set the Precharge to $\overline{\text{RAS}}$ delay (t_{RP})
- Set the $\overline{\text{RAS}}$ to Precharge delay (t_{RAS})
- Select the SDRAM power up mode (Init sequence)

SDRAM Enable

Bit0 should be set if there is an SDRAM in the system, otherwise it should be cleared. Any access to SDRAM while this bit is cleared causes a hardware error interrupt.

Selecting the CAS Latency Value (CL)

The $\overline{\text{CAS}}$ latency value defines the delay, in number of system clock cycles (SCLK), between the time that the SDRAM detects the read command and the time that it provides the data at its output pins. This parameter facilitates matching the SDRAM operation with the processor's ability to latch the data output.

$\overline{\text{CAS}}$ latency does not apply to write cycles.

The CL Bits2–1 in the `SDRCON` register select the $\overline{\text{CAS}}$ latency value as follows.

00 = 1 cycle latency

01 = 2 cycles latency

10 = 3 cycles latency

11 = Reserved

Generally, the frequency of the operation determines the value of the $\overline{\text{CAS}}$ latency.

Setting the SDRAM Buffering Option (Pipeline depth)

Systems that use several SDRAM devices connected in parallel may require buffering between the processor and multiple SDRAM devices in order to meet overall system timing requirements. To meet such timing requirements and enable intermediary buffering, the processor supports pipelining of SDRAM address and control signals. The pipeline depth Bit3 in the `SDRCON` register enables this mode:

Pipeline depth = 0 Disable pipelining

Pipeline depth = 1 Enable pipelining

When set (= 1), the SDRAM controller delays the data in write accesses by one cycle, enabling the processor to latch the address and controls externally. In read accesses, the SDRAM controller samples data one cycle later.

Figure 6-6 on page 6-24 shows another single processor example in which the SDRAM interface connects to multiple banks of SDRAM to provide 512M of SDRAM in a 4-bit I/O configuration. This configuration results in 16M x 32-bit words. In this example, 0xA and 0xB output from the registered buffers are the same signal, but are buffered separately. In the registered buffers, a delay of one clock cycle occurs between the input (Ix) and its corresponding output (0xA or 0xB).

SDRAM Programming

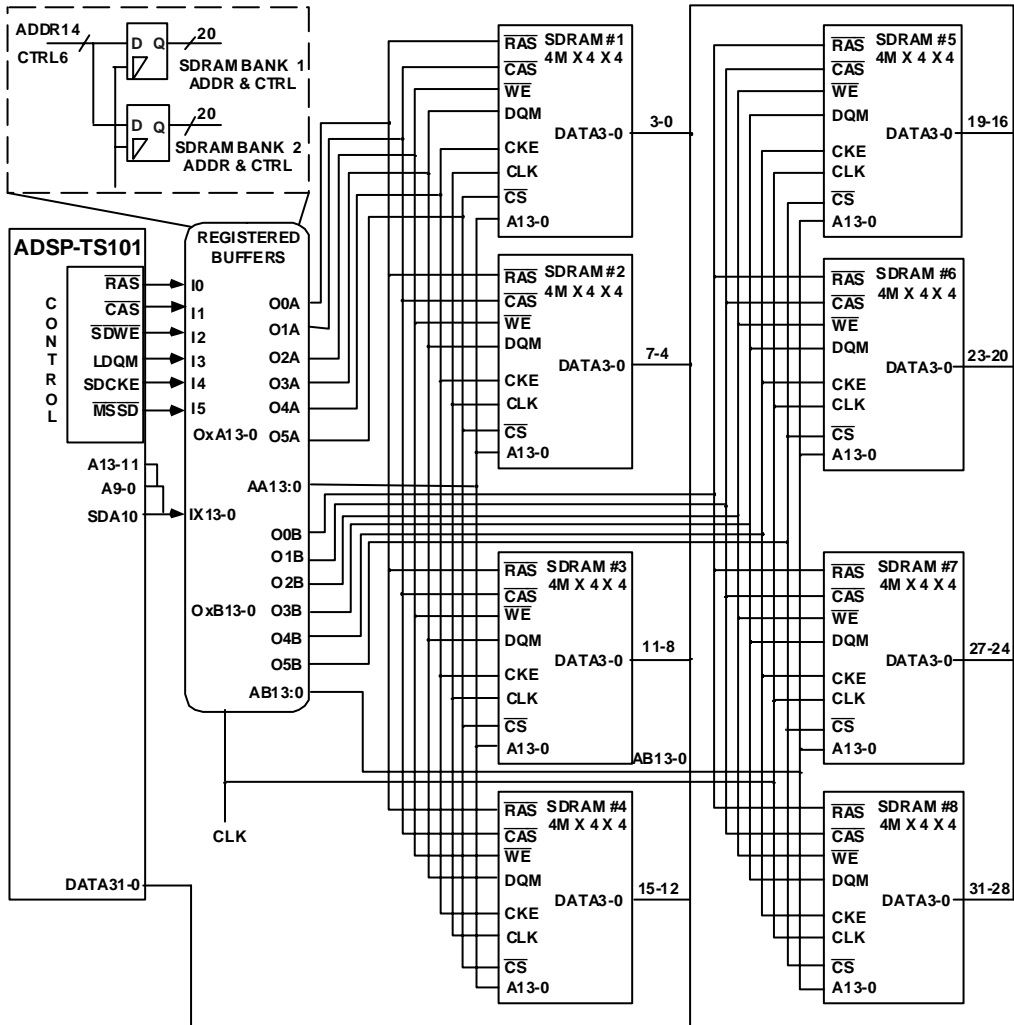


Figure 6-6. Uniprocessor System With Multiple SDRAM Devices (32-bit Bus)

Selecting the SDRAM Page Size (Page Boundary)

The processor's SDRAM controller page boundary bits define the page size, in number of words, of the SDRAM's banks. The setup of Bits5–4 in the SDRCON register is as follows.

00 = 256 words

01 = 512 words

10 = 1024 words

11 = Reserved

Setting the Refresh Counter Value (Refresh Rate)

Since the clock supplied to the SDRAM can vary, the processor provides a programmable refresh counter to coordinate the supplied clock rate with the SDRAM device's required refresh rate. Bits8–7 select between four different refresh rates calculated with the following equation:

$$f_{\text{cycles}} = \left(\text{clock} \times \frac{t_{\text{ref}}}{\text{rows}} \right) = (\text{clock} \times \text{refresh_rate})$$

00 = Once every 600 cycles

01 = Once every 900 cycles

10 = Once every 1200 cycles

11 = Once every 2400 cycles

Selecting the Precharge to RAS Delay (t_{RP})

The t_{RP} value (precharge to $\overline{\text{RAS}}$ delay) defines the required delay, in number of system clock cycles (SCLK), between the time the SDRAM controller issues a PRE command and the time it issues an ACT command.

SDRAM Programming

The setup of Bits10–9 in the `SDRCON` register for this parameter is shown in Table 6-8.

Table 6-8. Precharge to RAS Delay (t_{RP}) Bits

Delay	SDRCON Bits10–9
2 cycles	00
3 cycles	01
4 cycles	10
5 cycles	11

Selecting the RAS to Precharge Delay (t_{RAS})

The t_{RAS} value (\overline{RAS} to Precharge Delay) defines the required delay, in number of system clock cycles (`SCLK`), between the time the SDRAM controller issues an `ACT` command and the time it issues a `PRE` command.

The setup of Bits13–11 in the `SDRCON` register for this parameter is shown in Table 6-9.

Table 6-9. RAS to Precharge Delay (t_{RAS}) Bits

Delay	SDRCON Bits13–11
2 cycles	000
3 cycles	001
4 cycles	010
5 cycles	011
6 cycles	100
7 cycles	101
8 cycles	110

Setting the SDRAM Power-Up Mode (Init Sequence)

To avoid unpredictable start-up modes, SDRAM devices must follow a specific initialization sequence during power-up. The processor provides two commonly used power-up options.

The Init Sequence bit (14) in the `SDRCON` register selects the SDRAM power-up mode. When set (=1), the SDRAM controller sequentially issues: a `PRE` command, eight AutoRefresh cycles, and an `MRS` (Mode Register Set) command. When cleared (=0), the SDRAM controller issues, in this order: a `PRE` command, an `MRS` (Mode Register Set) command, and eight AutoRefresh cycles.

SDRAM Interface Throughput

Table 6-10 lists the data throughput rates for the processor's core or DMA read/write accesses to SDRAM. The following assumptions are made for the information in this table:

$\overline{\text{CAS}}$ latency = 2 cycles (CL = 2)

No SDRAM buffering (pipeline depth = 0)

Precharge (t_{RP}) = 2 cycles

Active command time (t_{RAS}) = 3 cycles

t_{RCD} (fixed) = CL = 2 cycles

Table 6-10. Data Throughput Rates ^{1,2}

Accesses	Operations	Page	Throughput per SDRAM Clock (64-bit words)
Sequential Uninterrupted	Read	Same	1 word/1 cycle
Non-sequential Uninterrupted	Read	Same	1 word/1 cycle

Multiprocessing Operation

Table 6-10. Data Throughput Rates (Cont'd)^{1,2}

Accesses	Operations	Page	Throughput per SDRAM Clock (64-bit words)
Sequential Interrupted	Read	Same	1 word/ 8 cycles (6 + CL)
Sequential Uninterrupted	Write	Same	1 word/1 cycle
Non-sequential Uninterrupted	Write	Same	1 word/1 cycle
Sequential Interrupted	Write	Same	1 word/1 cycle
Both	Read to Write	Same	1 word/6 cycles
Both	Write to Read	Same	1 word/4 cycles (2 + CL)
Non-sequential	Reads	Different	1 word/12 cycles (6 + t_{RP} + t_{RCD} + CL)
Non-sequential	Writes	Different	1 word/6 cycles
Autorefresh before read	Reads	Different	1 word/17 cycle 6 + 2(t_{RP} + t_{RAS} + t_{RCD} + CL)
Autorefresh before write	Writes	Different	1 word/11 cycle 2 + 2(t_{RP} + t_{RAS} + t_{RCD})

1 SYSCON external bus width configuration: 64-bit bus

2 With SDRAM buffering (pipeline depth = 1), replace any instance of (CL) with (CL+1)

Multiprocessing Operation

In a multiprocessing environment, the SDRAM is shared among two or more TigerSHARC processors. SDRAM input signals are always driven by the bus master. The slave processors track the commands that the master processor issues to the SDRAM. This feature or function helps to synchronize the SDRAM refresh counters and to prevent needless refreshing operations.

In multiprocessors systems, where the bus mastership changes, the current bus master automatically issues a precharge command (PRE) before the bus is relinquished to the new bus master. This way, the new bus master can safely start accessing the SDRAM by simply issuing an activation command (ACT).

In TigerSHARC multiprocessor systems where SDRAM is used, the Mode Register Set sequence is only issued by the TigerSHARC processor with ID = 000. Therefore, a TigerSHARC processor with ID = 000 must be present in every multiprocessor system.

Understanding DQM Operation

The HDQM/LDQM pins are used by the controller to mask write operations. HDQM three-states the SDRAM DQ buffers when performing 32-bit writes to even addresses in a 64-bit bus configuration, or when bus is configured as a 32-bit bus. LDQM three-states the SDRAM DQ buffers when performing writes to odd addresses in a 64-bit bus configuration. This data mask function does not apply for read operations, where the LDQM and HDQM pins are always low (inactive).

Powering Up After Reset

After reset, once the SDRCON register is written to in the user application code, the controller initiates the selected power up sequence. The exact sequence is determined by the Init Sequence bit in the SDRCON register. In a multiprocessing environment, the power up sequence is initiated by the TigerSHARC processor with ID = 000. Note that software reset does not reset the controller and does not re-initiate a power up sequence.

SDRAM Controller Commands

This section describes each command the SDRAM controller uses to manage the SDRAM interface. These commands are transparent to applications.

A summary of the various commands used by the on-chip controller for the SDRAM interface is as follows.

MRS (Mode Register Set). Initializes the SDRAM operation parameters during the power up sequence.

PRE (Precharge). Precharges the active bank.

ACT (Bank Activate). Activates a page in the required bank.

Read

Write

REF (Refresh). Causes the SDRAM to enter refresh mode and generate all addresses internally.

SREF (Self-Refresh). Places the SDRAM in self-refresh mode, in which it controls its refresh operations internally.

NOP (No Operation). Issued after a read or write to enable burst operation or to insert wait cycles for different SDRAM accesses. \overline{MSSD} is deasserted during this command.

Mode Register Set (MRS) Command

Mode Register Set (MRS) is a part of the system initialization sequence. MRS initializes SDRAM operation parameters by using ADDR13-0 of the SDRAM as data input. The MRS command is issued only by the TigerSHARC processor with ID = 000, and must be issued prior to the first SDRAM access after power up.

MRS consists of two optional sequences of which only one is selected, depending on the SDRCON Init Sequence bit. The availability of two optional types is to help support different SDRAM vendors.

MRS initializes the following parameters (using ADDR13-0):

Burst length – full page; Bits2–0; hardwired in the TigerSHARC processor

Wrap type – sequential; Bit3; hardwired in the TigerSHARC processor

Ltmode – latency mode ($\overline{\text{CAS}}$ latency); Bits6–4; set according to SDRCON programming

Bits13–7 – always 0; hardwired in the TigerSHARC processor

When executing MRS, the SDRAM must be in IDLE state in all its banks. The SDRAM cannot accept any other access during the two clock cycles following MRS. The SDRAM pin state during the MRS command is shown in Table 6-11 below.

Table 6-11. Pin State During MRS Command

Pin	State
$\overline{\text{MSSD}}$	low
$\overline{\text{CAS}}$	low
$\overline{\text{RAS}}$	low
$\overline{\text{SDWE}}$	low
SDCKE	high

Precharge (PRE) Command

The PRE command has two functions—terminate a read or write cycle and precharge the active bank.

SDRAM Controller Commands

Terminating Read/Write Cycles

The exact moment when the PRE command is issued to terminate a read or write cycle depends on the SDRAM latency mode.

Precharging

After power up a PRE command to all SDRAM banks is issued (SDA10 high). However, only the active bank precharges. Only one bank at a time can be active—no single bank precharge supported.

Precharge is issued in these cases:

1. During the SDRAM Init Sequence
2. Before ACT – access to a new page
3. Before refresh cycle
4. Before the TigerSHARC processor relinquishes the external bus



Following a PRE command, the SDRAM cannot accept any other access for the precharge to \overline{RAS} delay, t_{RP} . The appropriate field in the SDRCON register should be programmed according to the SDRAM characteristics.

The SDRAM pin state during the PRE command is shown in Table 6-12.

Table 6-12. Pin State During PRE Command

Pin	State
$\overline{\text{MSSD}}$	low
$\overline{\text{SDWE}}$	low
$\overline{\text{RAS}}$	low
$\overline{\text{CAS}}$	high
SDCKE	high
SDA10	high

Bank Active (ACT) Command

The ACT (Bank Active) command precedes any read or write access to a page that was not accessed beforehand. The ACT command is preceded with a PRE command. The ACT command asserts $\overline{\text{RAS}}$ and $\overline{\text{MSSD}}$ in order to enable the SDRAM to latch the row address. The ACT command opens up the SDRAM page, which remains open until the next precharge command. This is done to keep the page open even if there are occurrences of non-SDRAM accesses within a flow of accesses to a single SDRAM page.

Table 6-13. Pin State During ACT Command

Pin	State
$\overline{\text{MSSD}}$	low
$\overline{\text{CAS}}$	high
$\overline{\text{RAS}}$	low
$\overline{\text{SDWE}}$	high
SDCKE	high

Read Command

A Read command is preceded by an ACT command if it is the first access to the page. In Read commands, \overline{SDWE} is deasserted and \overline{CAS} and \overline{MSSD} are asserted to enable the SDRAM to latch the column address according to which burst start address is set.

The delay between ACT and Read commands is determined by the t_{RCD} parameter. Data is available after the t_{RCD} and \overline{CAS} latency requirements are met.

Single transactions take a different number of cycles according to transaction size and external bus width.

- Single-word read – 1 cycle
- Long read on 64-bit bus – 1 cycle
- Long read on 32-bit bus – 2 cycles
- Quad read on 64-bit bus – 2 cycles
- Quad read on 32-bit bus – 4 cycles

If no new command is issued to the SDRAM after a Read command, the TigerSHARC processor continues reading from the sequential addresses. This is called “page mode” or “burst”. If the whole transaction is more than one cycle (for example, quad transactions on a 32-bit bus), no new command is issued to the SDRAM and the rest of the data is read from sequential addresses. When the whole transaction is completed, the SDRAM can continue in different ways:

- If there is no new SDRAM transaction, the B_{stop} command is issued followed by a precharge command closing the active page.
- If there is an SDRAM read transaction to the same page, a new cycle begins in the next cycle.

- If there is an SDRAM write to the same page, B_{stop} command is issued and the write begins after a delay of \overline{CAS} latency to prevent contention on the data bus.
- If there is an SDRAM access to another page, B_{stop} command is issued and then, following the \overline{RAS} -to-precharge (t_{RP}) delay, Pre-charge (PRE) and Bank Active ACT cycles are issued.

Different examples of Read sequences are shown in Figure 6-7 on page 6-36 through Figure 6-8 on page 6-37.

Table 6-14. Pin State During Read Command

Pin	State
\overline{MSSD}	low
\overline{CAS}	low
\overline{RAS}	low
\overline{SDWE}	high
SDCKE	high

SDRAM Controller Commands

Bus Width = 64

For the sequence in Figure 6-7, the following is true:

- $\overline{\text{CAS}}$ latency = 2
- Bus width = 64

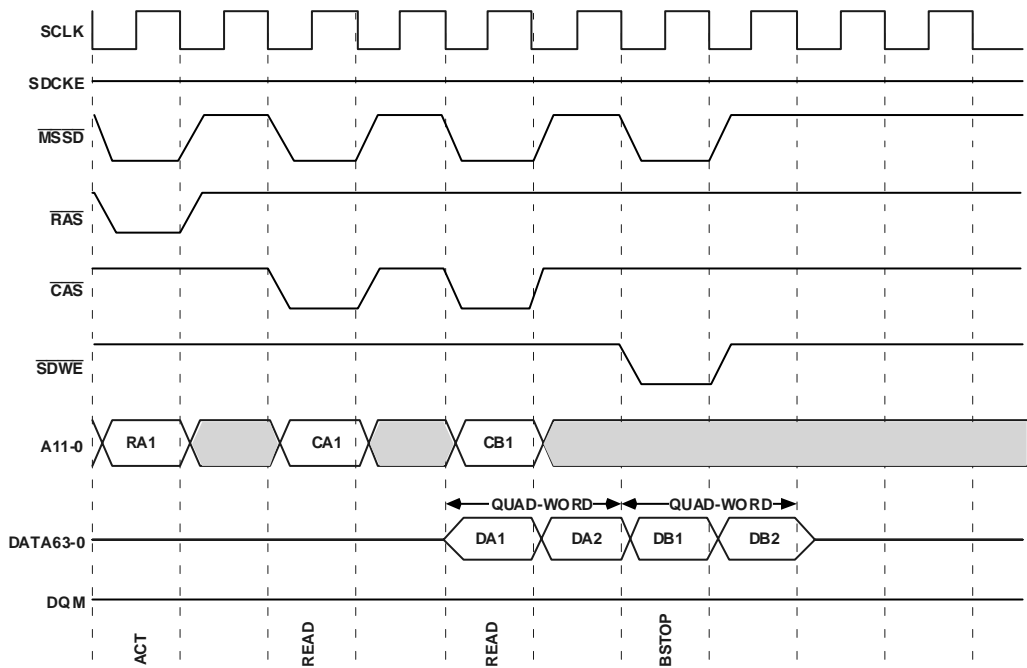


Figure 6-7. Bus Width = 64
(Burst Read Followed by Burst Read in the Same Page)

Bus Width = 32

For the sequence in Figure 6-8, the following is true:

- $\overline{\text{CAS}}$ latency = 2
- Bus width = 32

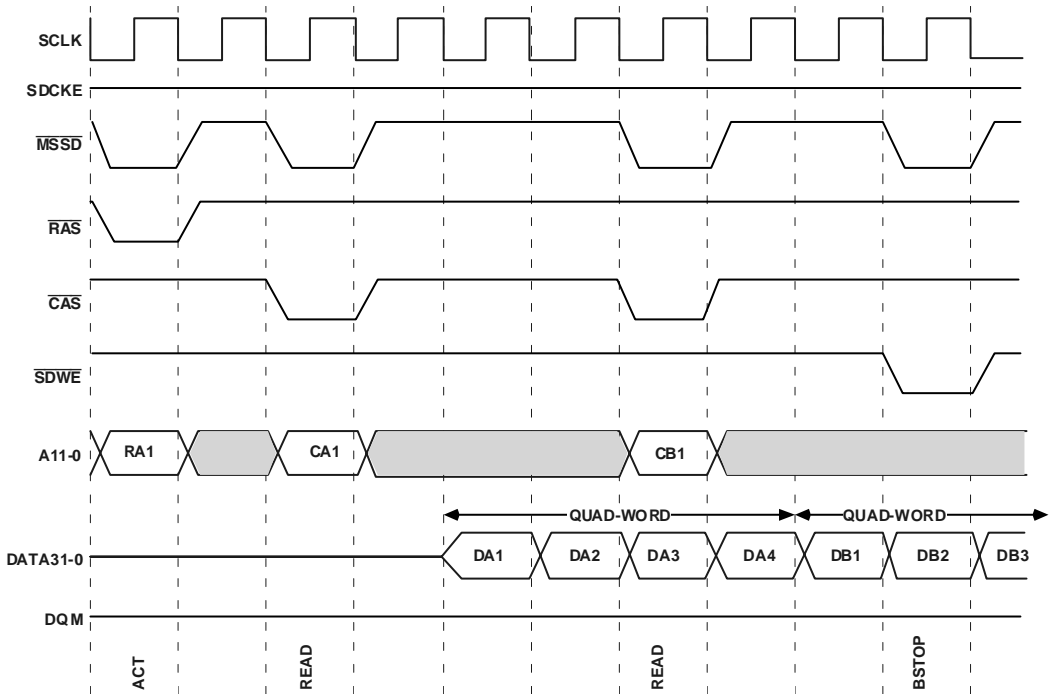


Figure 6-8. Bus Width = 32
(Burst Read Followed by Burst Read in the Same Page)

Write Command

A Write command is preceded by an ACT command if it is the first access to the page. In Write commands, $\overline{\text{CAS}}$, $\overline{\text{SDWE}}$, and $\overline{\text{MSSD}}$ are asserted to enable the SDRAM to latch the column address to set the burst start address.

The delay between ACT and Write commands is determined by the t_{RCD} parameter. Data is driven by the TigerSHARC processor on the Write command.

Single write transactions take a different number of cycles according to transaction size and external bus width.

- Single-word write – 1 cycle
- Long write on 64-bit bus – 1 cycle
- Long write on 32-bit bus – 2 cycles
- Quad write on 64-bit bus – 2 cycles
- Quad write on 32-bit bus – 4 cycles

If no new command is issued to the SDRAM after a Write command is issued, the SDRAM continues writing to sequential addresses. This is called ‘page mode’ or ‘burst’. If the whole transaction is more than one cycle—for example, quad write on a 32-bit bus—no new command is issued to the SDRAM and the rest of the data is written to sequential addresses. When the whole transaction is completed, the SDRAM can continue in different ways:

- If there is no new SDRAM transaction, Bstop command is issued, followed by a precharge command closing the active page.
- If there is an SDRAM write transaction to the same page, a new transaction begins on the next cycle.

- If there is an SDRAM read to the same page, B_{stop} command is issued to stop the write transaction, and the read transaction begins on the next cycle.
- If there is an SDRAM access to another page, B_{stop} command is issued and then, keeping to the RAS-to-precharge (t_{RP}) delay constraint, Precharge (PRE) and ACT cycles are issued.

Table 6-15. Pin State During Write Command

Pin	State
\overline{MSSD}	low
\overline{CAS}	low
\overline{RAS}	high
\overline{SDWE}	low
SDCKE	high

SDRAM Controller Commands

Bus Width = 64

For the sequence in Figure 6-10, the following is true:

- $\overline{\text{CAS}}$ latency = 2
- Bus width = 64

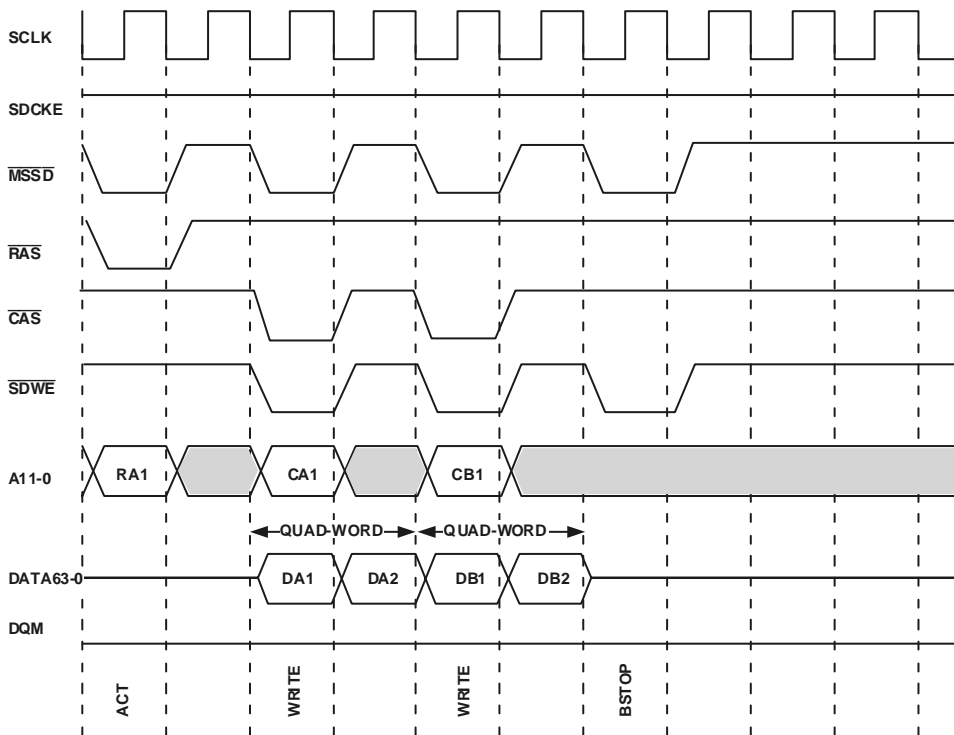


Figure 6-9. Bus Width = 64
(Burst Write Followed by Burst Write in the Same Page)

Bus Width = 32

For the sequence in Figure 6-10, the following is true:

- $\overline{\text{CAS}}$ latency = 2
- Bus width = 32

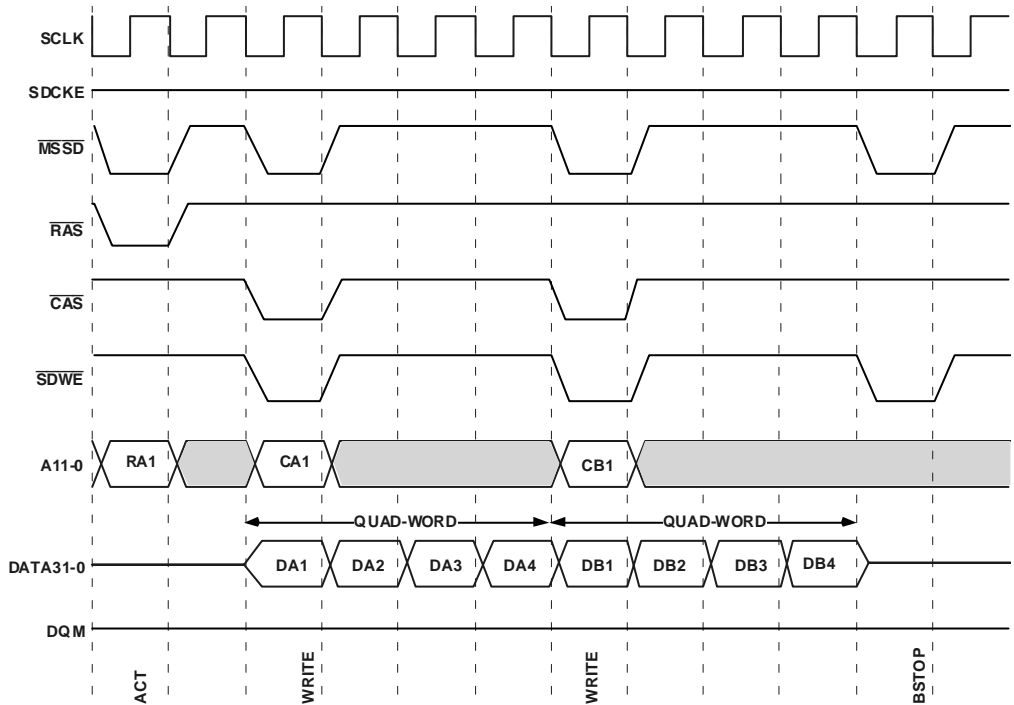


Figure 6-10. Bus Width = 32
(Burst Write Followed by Burst Write in the Same Page)

Refresh (REF) Command

The REF command is a request to the SDRAM to perform a CBR ($\overline{\text{CAS}}$ Before $\overline{\text{RAS}}$) refresh transaction. This transaction is generated automatically by the TigerSHARC processor and causes all addresses to be produced internally in the SDRAM. Before executing the REF command, the SDRAM controller executes a precharge (PRE) command to the active bank. The next active (ACT) command is given by the controller only after a minimum delay equal to t_{RC} .

The refresh cycle is required by the SDRAM to keep the data valid. The refresh frequency is set in the SDRCON register (see “SDRAM Control Register (SDRCON)” on page 6-19). In a multiprocessing system, the refresh is done by the current bus master. To eliminate problems caused by differences between TigerSHARC processors in refresh counter initialization, every slave TigerSHARC processor monitors the external bus and resets its own counter when identifying a refresh cycle.

Table 6-16. Pin State During REF Command

Pin	State
$\overline{\text{MSSD}}$	low
$\overline{\text{CAS}}$	low
$\overline{\text{RAS}}$	low
$\overline{\text{SDWE}}$	high
SDCKE	high

Self-Refresh (SREF) Command

The TigerSHARC processor enters self-refresh mode before the bus is relinquished to the host. The purpose is to keep the SDRAM self-refreshed in case the host doesn't interface with the SDRAM. If the host can interface with the SDRAM, it releases the SDRAM from

self-refresh mode and returns it to self-refresh mode before relinquishing the bus to the TigerSHARC processor. Before entering into self-refresh mode, the same conditions that apply to the refresh mode (see “Refresh (REF) Command” on page 6-42) must be sustained. Similarly, reaccessing the SDRAM after exiting from self-refresh mode is also limited by the same restrictions as for refresh mode. Self-refresh mode is entered by deasserting the $\overline{\text{SDCKE}}$ signal and is exited by reasserting the same signal.

The $\overline{\text{SREF}}$ command causes refresh operations to be performed internally by the SDRAM without any external control. Before executing the $\overline{\text{SREF}}$ command, the SDRAM precharges the active bank. After executing an $\overline{\text{SREF}}$ exit command, the controller waits $t_{\text{XSR}} = t_{\text{RC}} + t_{\text{RP}}$ to execute an auto refresh cycle. After, the auto refresh command, the SDRAM controller waits for t_{XSR} number of cycles before executing a bank active command (ACT).

Table 6-17. Pin State During REF Command

Pin	State
$\overline{\text{MSSD}}$	low
$\overline{\text{CAS}}$	low
$\overline{\text{RAS}}$	low
$\overline{\text{SDWE}}$	high
$\overline{\text{SDCKE}}$	low

Programming Example

This section provides a programming example written for the TigerSHARC processor. The example shown in Listing 6-1 on page 6-44 demonstrates how to set up the SDRAM controller to work with the ADSP-TS101 EZ-KIT Lite™.

Programming Example

Listing 6-1. SDRAM Controller Setup for ADSP-TS101 EZ-KIT Lite

```
#include <deffts101.h>
#include "TSEZkitDef.h"
/* ***** */
.section program;
SDRAM_Init:

xr0 = SYSCON_MP_WID64 | SYSCON_MEM_WID64 |
      SYSCON_MSH_SLOW | SYSCON_MSH_WT3   | SYSCON_MSH_IDLE |
      SYSCON_MS1_SLOW | SYSCON_MS1_WT3   | SYSCON_MS1_IDLE |
      SYSCON_MS0_SLOW | SYSCON_MS0_WT3   | SYSCON_MS0_IDLE ;;
SYSCON = xr0 ;;
/* SDRAM setting for 32Mb DIMM module and rev 1.3 EZ-Kit */
xr0 = SDRCON_INIT      | SDRCON_RAS2PC4 | SDRCON_PC2RAS3 |
      SDRCON_REF1200  | SDRCON_PG256   | SDRCON_CLAT2   |
SDRCON_ENBL ;;
SDRCON = xr0 ;;
```

7 DIRECT MEMORY ACCESS

Direct Memory Access (DMA) is a mechanism for transferring data without executing instructions in the processor core. The TigerSHARC processor on-chip DMA controller relieves the processor core of the burden of moving data between internal memory and an external device/memory, or between link ports and internal or external memory. The fully integrated DMA controller allows the TigerSHARC core processor, or an external device, to specify data transfer operations and return to normal processing while the DMA controller carries out the data transfers in the background.

The TigerSHARC processor DMA competes with other internal bus masters for internal memory access. For more information, see “Processor Microarchitecture” on page 5-3. This conflict is minimized because large internal memory bus bandwidth is available.

The TigerSHARC processor DMA includes 14 DMA channels of which four are dedicated to the transfer of data to and from external memory devices (including other TigerSHARC processors in the cluster), eight to the link ports and two to the AutoDMA registers. These DMA channels allow for the following transfer types (the <-> symbol indicates “to and from”)

- Internal memory <-> External memory or memory-mapped peripherals
- Internal memory <-> Internal memory of another TigerSHARC processor via the cluster bus
- Internal memory <-> Host processor

- Internal memory <--> Link port I/O
- External memory or memory-mapped peripherals <--> Link port I/O
- External memory <--> External peripherals
- Link port I/O <--> Link port I/O
- Cluster bus master <--> Slave internal memory via AutoDMA

Internal-to-internal memory transfers are not directly supported. Transfers may be executed through the multiprocessing space, although this loads the cluster bus.

The AutoDMA registers (see “AutoDMA Register Control” on page 7-33) are accessed by a cluster bus master or by the core itself via the multiprocessing address space. They cannot be accessed via the internal address space.

In chained DMA operations, a DMA transfer can be programmed to auto-initialize another DMA operation to follow the current one. This technique is primarily used for the same DMA channel, but in some cases, it may be used for a different channel.

The TigerSHARC processor also has four external DMA request pins ($\overline{\text{DMAR3-0}}$) that allow for external I/O devices to request DMA services. As a response to the DMA request, the TigerSHARC processor performs DMA transfers according to DMA channel initialization.

In order for a DMA channel to be initialized, the program must write to the DMA channel's Transfer Control Block (TCB). Figure 7-1 shows a block diagram of the DMA controller.

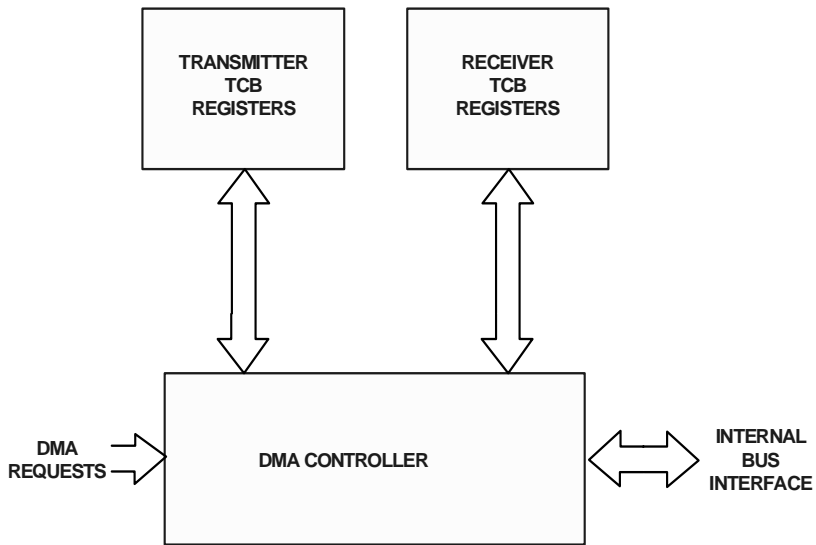


Figure 7-1. DMA Block Diagram

The Transfer Control Block is a quad-word (128-bit) register that contains the vital information required to perform a DMA. The form of the TCB register is shown in Figure 7-2 on page 7-4.

In the case of a transmitter TCB the four words contain the address of the source data, the number of words to be transferred, the address increment and the control bits.

In the case of a receiver TCB these four words contain destination address, the number of words to be received, the address increment and the control bits.



Figure 7-2. DMA TCB Register

The DI register is the 32-bit Index register for the DMA. This contains the source or destination of the data to be transmitted or received and can point to internal, external memory, or the link ports.

The DX register contains a 16-bit count value and a 16-bit modify value. The count value is stored in the upper 16 bits (16-31) and the modify in the lower (0-15). If a two-dimensional DMA is enabled, then this register contains the modify and count values for the X dimension only. The value of X count must always be the number of normal (32-bit) words to be transferred. Likewise, the modify value is the number of normal words to modify the count. For example, if we wanted to transmit four quad-words (16 normal words), then the count value would be 0x10 and the modify value 0x4 if the operand length in the DP register is set to quad-word. If the operand length was set to long-word, then the modify value would be 0x2. Programming the DMA TCB parameters is not limited to those as previously described. Figure 7-3 on page 7-6 shows some of the many options available for DMA operation.

i There are two restrictions that need to be considered when programming the DMA $TCBs$. The first is that the pointer must fall on an aligned boundary as specified by the operand length in the DP field. For example, it is not possible to set up a DMA TCB with an operand length of quad-word and a modify value of two as this results in a quad-word being transmitted from a non-quad-aligned

address and a software exception being generated. The second is that the count values entered into the DX and DY fields must be an integer multiple of the operand length. So if the operand length is programmed to long-word then the count values need to be a multiple of two.

The DY register is used in conjunction with the DX . This register contains the 16-bit modify and 16-bit count values for the DMA in the Y dimension. If two-dimensional DMA is not selected, then this register is not used and the contents are irrelevant.

The DP register contains all the control information for the DMA. This register is split into two main fields. The first contains all the control information and the second the chaining information. The breakdown of this register is shown in Figure 7-8 on page 7-18, Figure 7-9 on page 7-19, and Table 7-1 on page 7-20.

The cluster bus (EP) $TCBs$ are loaded by writing to the $DCSx$ registers for the source TCB and the $DCDx$ registers for the destination $TCBs$, where x can be any value ranging from 0 to 3. The DCx registers are written-to, in order to load the $TCBs$ for the link ports or the AutoDMA registers. The X can be any value from 4 to 13 depending on whether you are writing to link port receive, transmit, or AutoDMA channels.

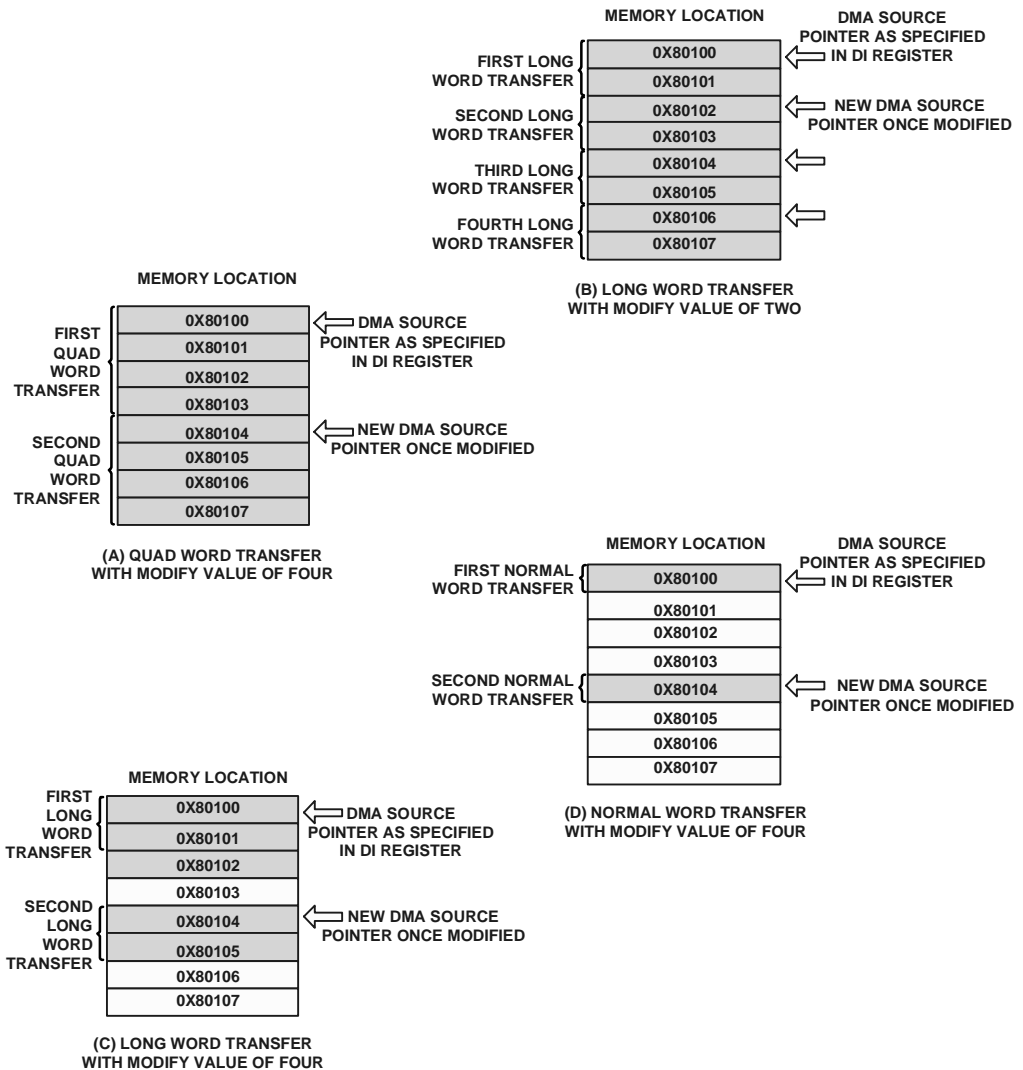


Figure 7-3. DMA Data Transfer Examples

DMA Controller Features

There are 14 DMA channels dedicated to six types of transfers:

Internal memory <=> Cluster bus	This transfer can be done in both directions and requires programming two Transfer Control Blocks (TCB). One is a transmitting TCB and the other a receiving TCB.
External memory <=> External I/O Device	This transfer can be done in both directions and requires programming two Transfer Control Blocks. One is a transmitting TCB and the other a receiving TCB.
Auto DMA register <=> Internal memory	This requires one receiver TCB.
Internal/external memory <=> Link ports	This requires one transmitter TCB.
Link ports <=> Internal/external memory	This requires one receiver TCB.
Link port <=> Link port	This requires one receiver TCB.

Cluster Bus Transfers

Cluster bus data transfers move data between the TigerSHARC processor internal memory and external memory or external I/O devices.

- Internal to external memory transfers

Transmitter DMA TCB registers are programmed with the internal memory address, the address increment, the number of transfers, and the control bits. Receiver DMA TCB registers are programmed with the external memory address, the address increment, the number of transfers, and the control bits.

DMA Controller Features

- External to internal memory transfers

Transmitter DMA TCB registers are programmed with the external memory address, the address increment, the number of transfers, and control bits. Receiver DMA TCB registers are programmed with the internal memory address, the address increment, the number of transfer, and control bits.

Once setup programming is complete, DMA transfers start automatically and either continue until the entire block is transferred, or until one transaction after each $\overline{\text{DMARX}}$ pulse. The programming of the TCBs determines the instructions to be followed.

- External device and external memory transfers

An additional DMA capability allows the TigerSHARC processor to support data transfers between an external device and external memory (flyby transactions). This transfer does not interfere with internal TigerSHARC processor operations.

An external I/O device, unlike memory, gives an indication of readiness for data transfer. A source device is ready if it has data to write. A receiver is ready when it has space in its write buffer. There are two techniques available for synchronizing these devices with the TigerSHARC processor DMA channels:

- The source or receiver can assert a DMA Request input ($\overline{\text{DMARX}}$) every time it is ready to transfer new data. The DMA, requests are accumulated in the DMA and a transaction is issued on the corresponding channel per DMA request. Up to 15 requests may be accumulated.
- A source that can be a master on the cluster bus can write to an AutoDMA register. After data is written to the AutoDMA register, the AutoDMA channel transfers the data according to its initialization.

AutoDMA Transfers

Receiver DMA TCB registers are programmed with the internal memory address, the address increment, the number of transfers, and the control bits.

Once setup is complete, when an external master (either a host or another TigerSHARC) writes to the AutoDMA register, a DMA request is initiated and the DMA channels transfers the data to the internal memory.

Link Transfers

Link DMA transfers handle data transmitted and received through the TigerSHARC processor's link ports. The following features are included.

- External/internal memory to link port transfers

Transmitter DMA TCB registers are programmed with the external/internal memory address, the address increment, the number of transfers, and the control bits.

Once transmitter TCB programming is complete, DMA transfers start automatically and continue until the block transfer is completed. The transmitting link port generates a DMA request if it is ready to receive data for transmission. The DMA from external/internal memory to the link port is then completed. This process is repeated until the entire data block is transferred.

- Link port to external/internal memory transfers

Receiver DMA TCB registers are programmed with the external/internal memory address, the address increment, the number of transfers, and the control bits.

DMA Controller Features

Once receiver TCB programming is completed, DMA transfers start automatically and the link port waits until data is received. The link port then generates a DMA request, and the DMA transfer to external/internal memory completes. This process continues until the block transfer is completed.

- Link port to external/internal memory transfers

Receiver DMA TCB registers are programmed with the external/internal memory address, the address increment, the number of transfers, and the control bits.

Once receiver TCB programming is completed, DMA transfers start automatically and the link port waits until data is received. The link port then generates a DMA request and the DMA transfer to external/internal memory completes. This process continues until the block transfer is completed.

Two-Dimensional DMA

The DMA is able to address and transfer two-dimensional memory arrays, where X and Y array dimensions are defined in transmitter and receiver TCB registers.

- Receiver array dimensions can differ from those of the transmitter, as long as the total number of words in source and destination are equal.
- A two-dimensional memory array can be moved to a one-dimensional array and vice versa, as long as the total number of words in source and destination are equal.
- A two-dimensional block in memory can be transmitted via links, or a block received through links or from AutoDMA can be placed in memory as a two-dimensional array.

Chained DMA

Chained DMA allows for the DMA controller to auto-initialize multiple DMA transfers, thus minimizing loading on the processor core. The chain pointer field of the DMA control bits points to the address in memory that contains the address, increment, number of transfers, and control bits for the next TCB to be loaded. The new TCB is loaded when the current DMA transfer completes.

The TigerSHARC DMA controller has a great flexibility of options with chained DMA sequences.

- Single channel chained DMA

When the chain pointer in the current TCB points to TCB data used to set up another DMA on the same channel.

- Cross channel chained DMA on link ports

When one link port DMA channel sets up a DMA transfer on a different link port DMA channel.

- Chain insertion

When a high priority DMA operation or another DMA chain may be inserted into an already active DMA chain.

DMA Architecture

DMA transfers are characterized by the direction of data flow, that is, from the transmitter (source) to the receiver (destination). If the transmitter or receiver is memory, it is characterized by a TCB register. Link and AutoDMA channels have one TCB register each. A transmit channel has one source TCB register. A receive link or AutoDMA channel has one desti-

DMA Architecture

nation TCB register. Feedthrough, from the receiver link to another transmitter link, can be caused by programming a receiver link channel TCB to send data to the target transmitter link buffer.

DMA initiates a transaction as specified by TCB programming and requests. The TCB programming determines if the DMA operates continuously or on requests (handshake mode). The link channels and AutoDMA channels always work by requests, which are generated internally by the link or by the AutoDMA registers. The memory-to-memory DMA channels can work either autonomously after initialization, or in handshake mode, using $\overline{\text{DMARx}}$ input pins. The bus transaction is always initiated by the DMA; the source of the transaction is the transmitter; and the transaction destination is the receiver.

Transactions that involve link or transactions where the transmitter is the internal memory are issued on the internal bus. Transactions where the transmitter is external memory are driven by the DMA directly to the BIU. For more information, see “Processor Microarchitecture” on page 5-3.

DMAR I/O Pins

The DMA request pins ($\overline{\text{DMAR3-0}}$) allow I/O devices to request DMA services from the TigerSHARC processor. As a response to a DMA request, the TigerSHARC processor performs DMA transfers according to DMA channels initialization. Any DMA request from an uninitialized channel is ignored. The $\overline{\text{DMAR}}$ inputs are edge-sensitive.

The TigerSHARC processor supports four external DMA request input signals, $\overline{\text{DMAR0}}$, $\overline{\text{DMAR1}}$, $\overline{\text{DMAR2}}$, and $\overline{\text{DMAR3}}$, and two output signals, $\overline{\text{FLYBY}}$ and $\overline{\text{TOEN}}$, to support DMA transfers between external peripheral devices. In Flyby transactions, internal memory or peripherals are not involved. Only channel 0 should be defined to work in Flyby mode.

By asserting a $\overline{\text{DMARx}}$ pin and waiting for the appropriate bus transaction to be performed by the TigerSHARC processor, an I/O device can transfer data to TigerSHARC processor internal memory or links. Any flyby output can be used to transfer data between an I/O device and external memory—in this case, the TigerSHARC processor performs a bus transaction but does not read or write data. “Handshake Mode” on page 7-61 provides additional information regarding DMAR I/O.

Terminology

Terms that appear several times in this chapter are defined below.

- External Port Input FIFO (IFIFO)

Refers to the TigerSHARC processor Bus Interface Unit (BIU) input FIFO. It is used for all externally-supplied data by bus masters (other TigerSHARC processors or a host processor), direct writes or external reads. The IFIFO also holds on-chip destination addresses and data attributes.

- External Port Output FIFO(OFIFO)

Refers to the TigerSHARC processor BIU output FIFO. It is used for all outgoing external port addresses, data, and transaction control signals, including DMA transfers to and from external address space.

- Transfer Control Block (TCB)

A quad-word that defines a set of parameters for the DMA operation.

- DMA TCB register

A quad-word register that contains a Transfer Control Block.

- TCB chain loading

Setting Up DMA Transfers

The process in which the TigerSHARC processor's DMA controller downloads a TCB from memory and autoinitializes the DMA TCB register.

- AutoDMA registers

Two virtual DMA registers that can only be accessed by a cluster bus master. These registers allow data to be moved to a block defined by the channel's TCB register. For more information, see "AutoDMA Register Control" on page 7-33.

Setting Up DMA Transfers

DMA operations can be programmed by the TigerSHARC processor core processor, by an external host processor, or by the (external) TigerSHARC processor bus master. The operation is programmed by writing to the memory-mapped DMA TCB registers. The TCB register is a quad-word register that indicates the DMA block transfer. A DMA channel is set up by writing a quad-word to each of the DMA TCB registers. Each register must be loaded with a starting address for the block, an address modifier, and a word count. Similar to link DMA channels, an AutoDMA register channel has only one TCB register.

In the case of link ports, once a DMA block is set up and enabled, data words received are automatically transferred to the internal memory or external memory or to another link port transmit buffer. Likewise, when the transmitting link is ready to transmit data, a quad-word is automatically transferred from internal or external memory to the transmitter link buffer. These transfers continue until the entire data buffer is received or transmitted.

If the TCB is programmed to issue an interrupt, then DMA interrupts are generated when an entire block of data has been transferred. This occurs when the DMA channel's count register has decremented to zero and the last element of data has been transferred.

DMA Transfer Control Block Registers

The TCB registers used to control and configure DMA operations are part of the memory-mapped register sets (“DMA Registers” on page 2-41). These can be accessed as quad-words only.

The remaining sections in this chapter describe the different operating modes of the DMA controller together with the associated control registers and bits. DMA TCB registers are detailed in the following subsections.

DMA Channel Control

Each of the four external memory DMA channels is controlled by a TCB quad-word register pair. In addition, each of the four link ports to or from memory DMA channels is controlled by a TCB quad-word register. Finally, each of the two AutoDMA registers-to-memory DMA channels is controlled by a TCB quad-word register.

Transfer Control Block (TCB) Registers

Each TCB register is 128 bits long and is divided into four 32-bit registers. These registers are illustrated in Figure 7-4.

- Index register (DI)
- X dimension count and increment register (DX)
- Y dimension count and increment register (DY)
- Control and chaining pointer (DP)

DMA Transfer Control Block Registers

These four registers form a quad-word TCB that can be loaded as an aligned quad-word from memory via chaining or by the core. The following conditions apply (note only quad accesses are allowed).

- Loading an inactive TCB (TY field in DP register = 000) into a TCB register reinitializes the state of the DMA channel, clearing the channel status bits to zero. The DMA request counters are flushed and any internal DMA states are reset.
- Loading an active TCB into an active channel TCB register generates an interrupt. The DSTAT register (see “DMA Status Register (DSTAT/DSTATC)” on page 7-23) can be read to determine if the channel is active.

To initiate a new master DMA sequence after the current one has finished, the program must write new parameters to the TCB registers, re-enabling the DMA (for chained DMA operations, this is done automatically.).

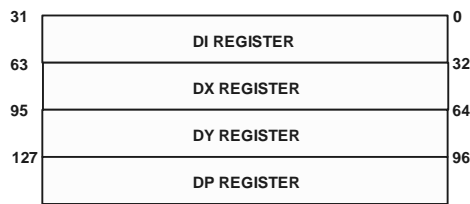


Figure 7-4. TCB Register

DIx Register

This is the 32-bit Index register for the DMA. It can point to the address of external, internal memory, or link ports.



Figure 7-5. DIx Register

DXx Register

If a two-dimensional (2D) DMA is disabled, this register contains the 16-bit modify (LSBs) and 16-bit count (MSBs) values for the DMA. If a two-dimensional (2D) DMA is enabled, it contains the 16-bit modify and 16-bit count X dimension values, where X count is the number of normal words to be transferred, and X modify is the number of normal words to modify the address pointer.

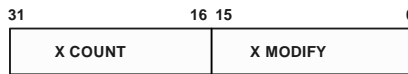


Figure 7-6. DXx Register

DYx Register

This register is used together with the DX register when a two-dimensional DMA is enabled. It contains the 16-bit modify and the 16-bit Y count dimension values, where Y modify is the difference between the address of the last data element in a row and the address of the first element in the following row. If a two-dimensional (2D) DMA is disabled, this register is not used and any contents loaded are ignored.

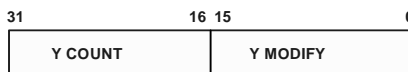


Figure 7-7. DYx Register

DMA Transfer Control Block Registers

DPx Register

This register is split into two fields, where the first is dedicated to DMA control and the second is dedicated to chaining.

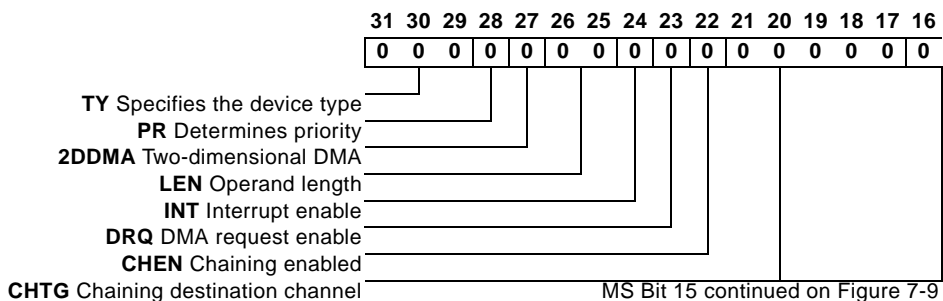


Figure 7-8. DPx (Upper) Register Bit Descriptions

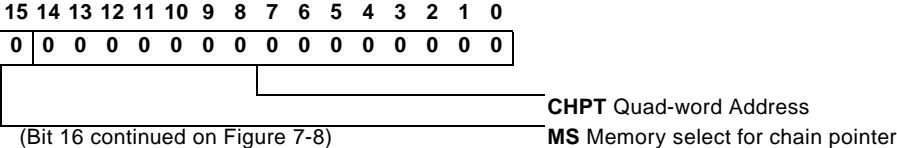


Figure 7-9. DPx (Lower) Register Bit Descriptions

DMA Transfer Control Block Registers

Table 7-1. DPx Register Bit Descriptions

Bit	Description
CHPT Bits14–0	Chain Pointer These bits contain Bits16–2 of the address in memory where the next TCB register contents are to be found. Bits1–0 of the memory address are not required as the address must be quad-word aligned. For example if the next TCB register contents are stored at memory location 0x87128-0x8712B then the CHPT field would contain b00100001110001001010 (0x21C4A).
MS Bits16–15	Memory select for chain pointer These bits specify the memory block in which the next TCB contents can be found. 00 – Memory Block 0 01 – Memory Block 1 10 – Memory Block 2 11 – Reserved
CHTG Bits21–17 ¹	Chaining destination channel This field specifies the DMA channel TCB registers in which the quad-word pointed to by the CHPT and MS fields should be loaded. 00000 – Channel 0 Source TCB 00001 – Channel 0 Destination TCB 00010 – Channel 1 Source TCB 00011 – Channel 1 Destination TCB 00100 – Channel 2 Source TCB 00101 – Channel 2 Destination TCB 00110 – Channel 3 Source TCB 00111 – Channel 3 Destination TCB 01000 – Channel 4 Link Port 0 Transmit TCB 01001 – Channel 5 Link Port 1 Transmit TCB 01010 – Channel 6 Link Port 2 Transmit TCB 01011 – Channel 7 Link Port 3 Transmit TCB 10000 – Channel 8 Link Port 0 Receive TCB 10001 – Channel 9 Link Port 1 Receive TCB 10010 – Channel 10 Link Port 2 Receive TCB 10011 – Channel 11 Link Port 3 Receive TCB 10110 – Channel 12 IFIFO AutoDMA0 Receive TCB 10111 – Channel 13 IFIFO AutoDMA1 Receive TCB

Table 7-1. DPx Register Bit Descriptions (Cont'd)

Bit	Description
CHEN Bit22	<p>Chaining Enabled</p> <p>0 – No Chaining</p> <p>1 – DMA loads chaining targets channel TCB registers from internal memory</p> <p>This bit disables chaining or enables chaining. If set the DMA loads the chaining destination channel TCB registers from the internal memory location pointed to by the CHPT and MS fields.</p>
DRQ Bit23	<p>DMA Request Enable</p> <p>0 – Once the DMA channel is enabled, the whole block is transferred</p> <p>1 – DMA issues transactions only upon request</p> <p>If cleared (=0) this bit disables the functionality of the $\overline{\text{DMARX}}$ pins on external port DMA transfers. This results in the entire block of data being transferred once the TCB registers have been written to.</p> <p>If enabled (=1) this bit results in external port transactions occurring only upon a request signaled on the $\overline{\text{DMARX}}$ pins.</p> <p>For AutoDMA channels this bit must always be set.</p>
INT Bit24	<p>Interrupt Enable</p> <p>0 – DMA interrupt is disabled</p> <p>1 – The DMA interrupts the core after transferring the whole block</p> <p>If cleared (=0) this bit disables the generation of an interrupt on completion of the DMA block transfer.</p> <p>When enabled (=1) an interrupt is generated upon completion of the DMA. The interrupt is generated when the count field of the DMA channels TCB decrements to zero.</p>
LEN Bits26–25	<p>Operand Length</p> <p>00 – Reserved</p> <p>01 – Normal (32-bit) word</p> <p>10 – Long (64-bit) word</p> <p>11 – Quad (128-bit) word</p> <p>These bits specify the length of the operand to be transferred on each DMA transaction. The original source or destination address specified in the DIx register must be aligned to a word boundary as specified with the setting of these bits. Thus if operand length is set to quad, then the address originally loaded in the DIx register must be divisible by four.</p>

DMA Transfer Control Block Registers

Table 7-1. DPx Register Bit Descriptions (Cont'd)

Bit	Description
2DDMA Bit27	<p>Two-Dimensional DMA</p> <p>0 – One-dimensional DMA: DY register is meaningless</p> <p>1 – Two-dimensional DMA: DX controls X dimension addresses, increment, and count. The DY register controls Y dimension addresses, increment, and count. In this mode of operation, the DX register contains the X dimension addresses, increment and count. The DY register controls the Y dimension addresses, increment, and count. When disabled (cleared = 0), the contents of the DY register are simply ignored, and a normal one-dimensional DMA is performed.</p>
PR Bit28	<p>Determines Priority</p> <p>0 – DMA request priority is normal</p> <p>1 – DMA request priority is high</p> <p>This bit determines the priority of the DMA. When set (PR = 1), the DMA request is given a high priority. If the TCB refers to an external address, then the Priority Access (DPA) pin is set. See cluster bus definition for more information on the DPA pin. If the TCB refers to an internal address, the DMA asserts its high priority internal bus request. When the priority bit is cleared (PR=0), the DMA request priority is normal. This bit also indicates the priority in the DMA channel arbitration.</p>
TY Bits31–29	<p>Specify the Device Type</p> <p>000 – DMA disabled</p> <p>001 – I/O link port DMA</p> <p>This type is set when performing link port to link port transfers.</p> <p>010 – Internal memory</p> <p>This type must be set in the source TCB if the source is internal memory or if the destination of the transfer is internal memory.</p> <p>011 – Reserved</p> <p>100 – External memory</p> <p>The transmitter and receiver TCB registers should be programmed to this type if the source or destination is in external memory space.</p> <p>101 – External I/O device (Flyby)</p> <p>110 – Boot EPROM This type allows for DMA accesses to the boot EPROM memory. The BMS pin is used to select the device when this type is set.</p> <p>111 – Reserved</p>

- 1 Link channel TCBs CHTG field may be programmed with any other link channel. However, CHTG should not be cross chained with any other DMA channels.

DMA Control and Status Registers

The following registers act as status and control registers for the DMA:

- DMA Status Register – `DSTAT`
- DMA Control Registers – `DCNT`, `DCNTST`, `DCNTCL`

DMA Status Register (DSTAT/DSTATC)

The `DSTAT` register allocates a field of three status bits per channel, indicating whether the channel is currently disabled or active; whether the DMA has completed or not; and whether an error has been detected. The bits are read only and a field is cleared by loading the respective `TCB` register.

- Active status is set when the DMA channel is enabled (when the `TY` field of the `DPx` register is set).
- DMA completion status is set when the last DMA transaction is completed.
- Error status is set if an illegal or error condition is detected.

There are three DMA channel status states in which a hardware interrupt may be generated. These states are reached if one or more of the following conditions occur:

- Writing to already active DMA channels `TCB` registers
- Writing an illegal configuration to DMA channels `TCB` registers (`DP` register)
- Initializing a DMA channel to read from broadcast memory space

If any of these conditions occur a hardware interrupt is generated if enabled and the status bits can only be cleared by reading from the `DSTATC` register.

DMA Control and Status Registers

The `DSTAT` and `DSTATC` registers must be read as either a long or quad-word. Normal word reading of these registers is not permitted.

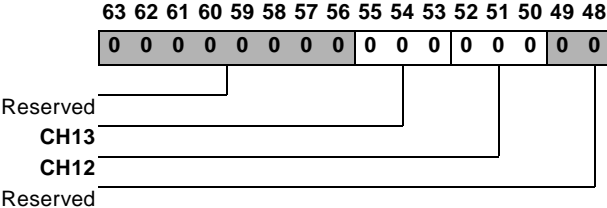
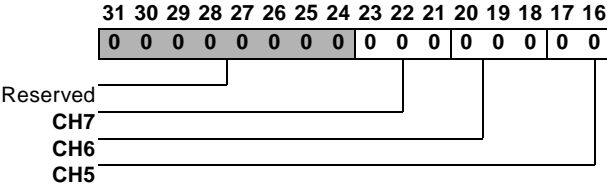


Figure 7-10. `DSTAT` (Bits 63-48) Register Bit Descriptions



Bits 17-15 continued on Figure 7-13

Figure 7-11. `DSTAT` (Bits 31-16) Register Bit Descriptions

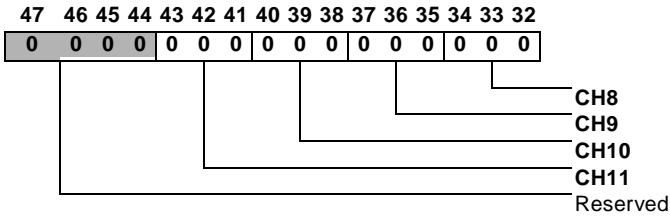


Figure 7-12. DSTAT (Bits 47-32) Register Bit Descriptions

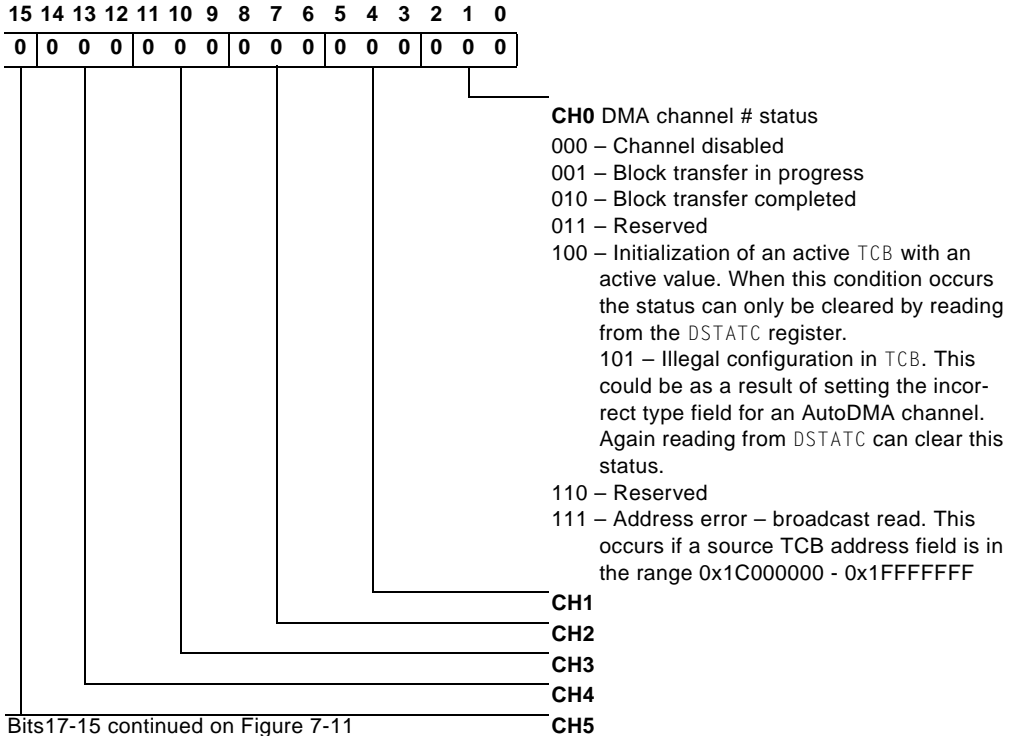


Figure 7-13. DSTAT (Bits 15-0) Register Bit Descriptions

DMA Control and Status Registers

DMA Control Registers

There are three DMA control registers—DCNT, DCNTST, and DCNTCL.

DCNT Register

This 32-bit control register controls the DMA flow. Setting DCNT bits stop DMA flow at the end of the current transaction; clearing DCNT bits resumes a suspended transaction. Initial value of the DCNT after reset is 0x0.

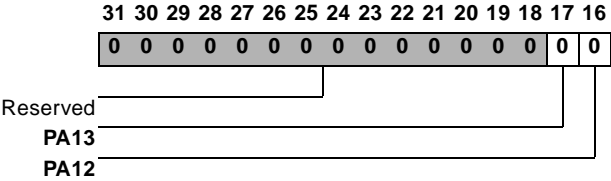


Figure 7-14. DCNT (Upper) Register

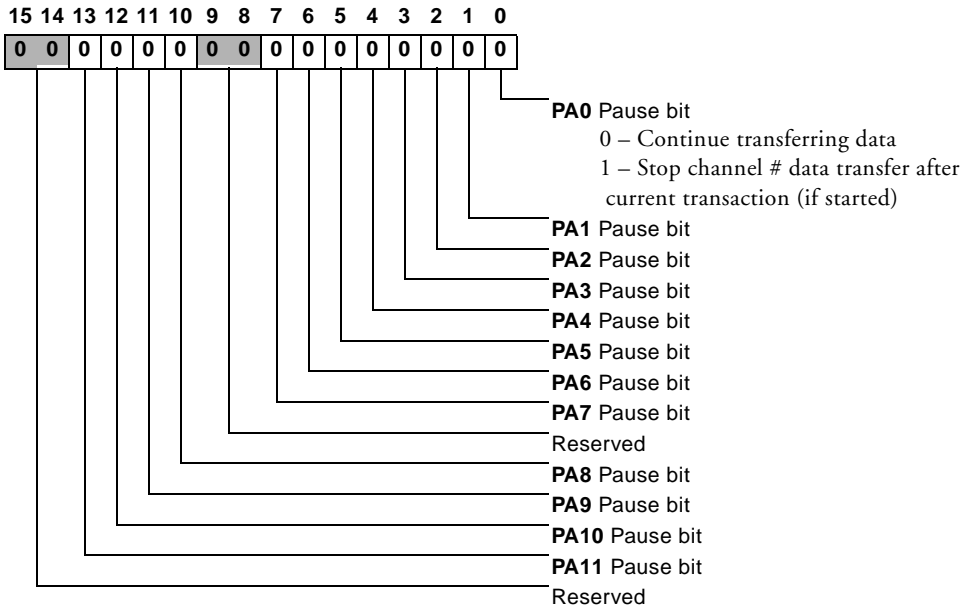


Figure 7-15. DCNT (Lower) Register

DMA Control and Status Registers

DCNTST Register

DCNTST is a 32-bit register alias used to set DCNT bits. The value written to DCNTST is ORed with DCNT, and the result is loaded into DCNT. For example, writing a 1 to bit n in the register sets the corresponding bit n in the DCNT register.

DCNTCL Register

DCNTCL is a 32-bit register alias used to clear DCNT bits. The value written to DCNTCL is ANDed with DCNT, and the result is loaded into DCNT. For example, writing a 0 to bit n in the register resets the corresponding bit n in the DCNT register.

DMA Control Register Restrictions

The following restrictions apply to the DMA Control registers.

Operand Length Setup (Len)

In external port (EP) channels, the LEN field (normal word, long-word or quad-word) must be the same in both TCBs. If the TY setup of either source or destination TCB is boot EPROM, the LEN field must be 0x1 (normal word).

In link (receive and transmit) channels the LEN field must always be quad-word (0x3). For AutoDMA channels, the LEN field must be the same as the transactions to the AutoDMA register.

Count (XCOUNT and YCOUNT)

For EP channels, the total counts should be identical.

- DX_COUNT: when the 2DDMA bit is cleared
- DX_COUNT * DY_COUNT: when the 2DDMA bit is set

The 2DDMA bit may be set in one of the TCBS and cleared in the other.

There is no restriction on count registers in link DMA channels. In all cases, if a channel is not idle, the DX_COUNT cannot be zero. If 2DDMA is set, DY_COUNT cannot be 0 or 1.

Type Setup – Links Transmit (Channels 4 to 7)

For DMA control registers, link transmit channels can be set up as only one of the following:

- 2 (internal memory)
- 4 (external memory)

Type Setup – Links Receive (Channels 8 to 11)

For DMA control registers, link receive channels can be set up as only one of the following:

- 1 (link port),
- 2 (internal memory), or
- 4 (external memory)

DMA Control and Status Registers

Type Setup – EP (Channels 0 to 3)

Table 7-2. EP Channels (0 to 3) Type Restrictions

Source Type	Destination Type
2 (Internal Memory)	4 (External Memory) or 6 (Boot EPROM)
4 (External Memory)	2 (Internal Memory) or 5 (External I/O Flyby)
5 (External I/O Flyby)	4 (External Memory)
6 (Boot EPROM)	2 (Internal Memory)
0 (Channel Disabled)	Any
Any	0 (Channel Disabled)

Type Setup – AutoDMA (Channels 12, 13)

AutoDMA is limited to internal memory. The type must be 2 (internal memory).

DMA Request

In both AutoDMA channels, the DRQ bit (DMA Request) must be set.

In EP channels (channels 0 to 3), the DRQ bit must be the same in both TCB registers.

Alignments

If the LEN field is 2 (long-word), the following fields must be even (bit 0 cleared):

- DI
- DX_MODIFY

- `DY_MODIFY` (if relevant)
- `DX_COUNT`

If the `LEN` field is 3 (quad-word), the values must all be multiplicands of 4 (Bits1–0 cleared).

For type = 6 (boot EPROM), these registers must all be multiplicands of 4 (Bits1–0 cleared).

Address Range

The following correlation must exist between the `TY` field in the `DP` register and the address any time the `TCB` is active:

- `TY = 1` (link) \Rightarrow address is of one of the link transmit registers (see “Link Registers” on page 2-47).
- `TY = 2` (internal memory) \Rightarrow address is in internal memory range, and not in the registers (see “Internal Address Space” on page 2-6).
- `TY = 4` (external memory) \Rightarrow address is in external memory range (address31–22 non-zero). It can be in host, external memory or multiprocessing space.



When a link DMA channel (transmit or receive channel) becomes active, if the transmit buffer is empty (for transmit channel) or if the receive buffer is not empty (for receive channel), a DMA request is immediately issued.

DMA Controller Operations

Link Port DMA Control

The TigerSHARC processor's four link ports are able to use DMA transfers to handle transmit and receive data. Two DMA channels are assigned to each link port, where the link port DMA request specifies the DMA channel that serves them.

- The link port input channel has a receiver TCB register.
- The link port output channel has a transmitter TCB register.


The number of words to be transferred is defined as X count or X count times Y count (if 2DDMA is set), but in any case, only quad-word transfers are allowed. The link DMA channel is enabled as long as the TY field is not set to 000 in the TCB register DP. Control bits and chaining address are as defined in TCB register DP, where if chaining is enabled, one quad-word TCB register is loaded. The link's functionality is specified in “Link Ports” on page 8-1. Finally, DMA request priority is fixed—link 3 is highest and link 0 is the lowest.

External Port DMA Control

The TigerSHARC processor's external port is able to use DMA transfers to transmit and receive data. Four DMA channels are assigned to external port operations. Each channel can work in handshake mode, using its $\overline{\text{DMARx}}$ pin. Either 32-bit (word), 64-bit (long-word), or 128-bit (quad-word) internal data widths can be used when transferring data between internal and external memory.

AutoDMA Register Control

The AutoDMA is a technique used to implement a slave mode DMA. There are two virtual registers—AutoDMA0 and AutoDMA1. Each of the AutoDMAs has an AutoDMA channel associated with it, channels 12 and 13, respectively. When an external master (host or another TigerSHARC processor) writes to one of the AutoDMA registers, this register issues a DMA request to its associated channel and the DMA channel transfers the data to internal memory according to its TCB programming. Either 32-, 64- or 128-bit internal data width can be used when transferring data between the data registers and internal memory. The DMA data registers channel is enabled as long as the TY field is not set to 000 in the TCB register DP. Control bits and chaining address are as defined in TCB register DP. If chaining is enabled, one TCB register is loaded.

 If you write to AutoDMA while it is not initialized, the written data is lost, and an error is indicated in the SYSTAT register. Refer to the “SYSTAT/SYSTATCL Register” on page 2-31 for additional information.

DMA Transfers

The following subsections discuss the operation of the TigerSHARC processor’s DMA controller and DMA transfers.

Internal Memory Buses

DMA controller operations are carried out on any one of the three internal memory buses. All I/O ports (link and external) are connected to the internal memory via the three memory buses. For more information, see “Processor Microarchitecture” on page 5-3. The DMA controller generates an internal memory access on one of these buses.

DMA Controller Operations

DMA Channels

Memory-to-memory DMA channels consist of two TCB registers (one for receive and one for transmit) that specify:

- A data buffer in internal and external memory
- Control fields
- The method of requesting a transfer

Link memory DMA channels consist of one TCB register that also specifies a data buffer in internal or external memory, control fields, and the hardware required to request DMA service. Any receiver link channel TCB register can define another link port as a memory-mapped device.

AutoDMA register memory is similar to link memory except that AutoDMA registers are always receivers. The AutoDMA register memory TCB cannot define another link port as a memory-mapped device. The destination must always be to internal memory.

The DMA controller contains priority logic to determine which channel or I/O can drive which bus in any given cycle.


DMA Memory Accesses

The DMA is able to access the full address space. Each channel includes an Index register (DI_x) and a Modify register (DM_x) in its TCB register, which are used to set up a data buffer in the memory.


The 32-bit TCB register DI holds the block's initial address and must be initialized with a starting address for the data buffer. The address in the DI register is directed to one of the internal memories, to a link register address, or to the external memory space. This must be consistent with the TCB transfer type—TY field in DP_x register. (See “DP_x Register” on page 7-18.)

The data transferred may be a normal, long, or quad-word. This is specified in the `LEN` field of the `TCB` control register `DPx`. (See “`DPx` Register” on page 7-18.) The `LEN` field must be unique per channel. (Both external port channel’s `TCBs` must have the same `LEN` value.)

After each data transfer to or from internal memory, the DMA controller adds the modify value to the Index register to generate the address for the next DMA transfer. The modify value is added to the index value and written back into the Index register. The modify value in the X modify field in the `TCB` register `DX` is a 16-bit signed integer, allowing both increments and decrements. Similarly the Y modify field in the `TCB` register `DY` is a 16-bit signed integer. The 16 bits are sign-extended when used with the 32-bit Index register.

-  If the Index register is modified out of the address range of an existing memory allocation or changes range between internal and external memory space, results are unpredictable.

Each DMA channel has a Count register (the X count field in the `TCB` register `DX`) that has to be initialized with the number of words to be transferred, regardless of the data item length (32, 64 or 128 bits). The count register is decremented after each DMA transfer on that channel. The decrement value is 1, 2, or 4 depending on the normal, long, or quad-word access specified. When the count reaches zero, the DMA is complete, and the interrupt for that channel can be generated.

-  If the X count field in the `TCB` register `DX` is initialized with zero, DMA transfers on that channel are not disabled. Moreover, 2^{16} transfers are performed. This occurs because the first transfer is initiated before the count value is tested. The correct way to disable a DMA channel is to clear its DMA enable bit in the corresponding control register.

Each DMA channel `TCB` also has a chain pointer—the `CHPT` field in the `TCB` register `DP`. The chain pointer is used in chained DMA operations. (See “DMA Chaining” on page 7-41.)

DMA Controller Operations

When DMA is from internal/external memory to the external/internal memory, two TCB registers are used. One is used for the internal DMA address generation; the second is used to generate 32-bit addresses to be driven out of the external port. If the two-dimensional DMA option is enabled, the count value equals the TCB register $DX \times$ count times the TCB register $DY \times$ count. It is important to stress that different count values cause the DMA to cease once the transfer of the shorter block is complete.

“Transfer Control Block (TCB) Registers” on page 7-15 lists the TCB registers for each DMA channel. The TCB registers are disabled following a hardware reset, except for the boot channel TCB. For more information, see “DMA Operation on Boot” on page 10-32.

DMA Channel Prioritization

The overall DMA prioritization is defined by internal bus priority as well as DMA channels priority.

Internal Memory Bus Priority

DMA arbitration resolves priority conflicts between DMA channels. Internal memory buses have different arbitration levels that define priority between internal memory bus masters. The internal memory bus priority is described in “Processor Microarchitecture” on page 5-3.

DMA Channel Priority

Since more than one DMA channel may have a request active in a particular cycle, a prioritization scheme is used to select the channel to service. The TigerSHARC processor always uses a fixed prioritization between I/O groups. Table 7-3 on page 7-38 lists the DMA groups in descending order of priority.

Priority is also set according to these considerations:

- Incoming data has higher priority than outgoing data, in order to reduce the possibility of stalling the cluster bus. For this reason, the AutoDMA register channels are assigned the highest priority. Receiving links have priority over transmitting links.
- The links have higher priority than the external ports because their lower bandwidth is not likely to block the bus.
- External port channels have a rotating priority. The sequence in which they rotate is 3-2-1-0.
- If the PR bit is set in TCB_x register DP, channel *x* asserts the highest priority inside its group. If more than one channel has one of its PR bits set in the TCB register DP, priority is resolved according to the priority list.

DMA Controller Operations

- The DMA controller determines the highest priority channel/device(s) that is requesting during every cycle. If the DMA is transferring a block and a higher priority channel requests DMA services, arbitration is carried out and the transfer is stopped.
- TCB chain loading receives the channel's priority. For example, a higher priority channel data transfer is performed before a lower priority TCB chain loading.

Table 7-3. DMA Channel Priority

AutoDMA Registers	Highest Priority
Channel 13	↓
Channel 12	↓
Receiving Links	↓
Channel 11 (Link 3)	↓
Channel 10 (Link 2)	↓
Channel 9 (Link 1)	↓
Channel 8 (Link 0)	↓
Transmitting Links	↓
Channel 7 (Link 3)	↓
Channel 6 (Link 2)	↓
Channel 5 (Link 1)	↓
Channel 4 (Link 0)	↓
External Ports (EPs)	↓
Channel 3 ($\overline{\text{DMAR3}}$)	↓
Channel 2 ($\overline{\text{DMAR2}}$)	↓
Channel 1 ($\overline{\text{DMAR1}}$)	↓
Channel 0 ($\overline{\text{DMAR0}}$)	Lowest Priority

Rotating Priority

The priority of channels 0 to 3 rotates—that is, every time a new selection is made, the priority is set again. The priority of the channels after reset is:

- channel 3 (highest)
- channel 2
- channel 1
- channel 0 (lowest)

Every time a new selection is made, the priority changes. The newly selected channel receives the highest priority, where priority descends in the same order as above, but in a cyclical fashion. For example, if channel 1 becomes the next selected channel, the priority is:

- channel 1 (highest)
- channel 0
- channel 3
- channel 2 (lowest)

If the next selected channel is channel 0, the priority scheme is:

- channel 0 (highest)
- channel 3
- channel 2
- channel 1 (lowest)

The priority within channels 0 to 3 is kept while higher priority channels (one or more of channels 4–13) are selected. When the higher priority channel (or channels) completes its transfer, the last priority order for channels 0 to 3 is kept unchanged.

DMA Controller Operations

This is under the assumption that all channels do not set the priority bit in TCB, or that all channels do set it. If one or more of the channels is set to high priority, it is selected at a separate arbitration level. In this case there is one round robin for the higher priority channels, and another round robin for the lower priority channels. For example, if after reset channel 2, the TCB priority bit is set, the priority is:

- channel 2 (priority bit set)
- channel 3 (highest)
- channel 1
- channel 0 (lowest)

If channel 1 requests and the request is granted, the priority scheme is:

- channel 2 (priority bit set)
- channel 1 (highest)
- channel 0
- channel 3 (lowest)


If during the operation of channel 1, channel 2 makes a request, it is granted and the channel 1 operation is interrupted. When it completes executing its transactions, the scheme of the lower priority channels is unchanged, and channel 1 wins the arbitration and continues its transfers.

If there is more than one channel with priority bit set, the arbitration between these channels is in a similar round robin manner.

DMA Chaining

DMA chaining allows the TigerSHARC processor's DMA controller to auto-initialize itself between multiple DMA transfers. Using chaining, multiple DMA operations can be set up, in which each operation can have different attributes and I/Os.

There are some restrictions on chaining DMA across DMA channels. For internal/external memory DMA transfers, cross channel DMA is not permitted. DMA processes between internal/external memory can only be chained within the same channel. Link port DMA processes, however, may be chained across channels.

-  If chaining across link port DMA channels, the system should enable the DMA error interrupt. This interrupt should be enabled to detect cross-channel chaining to an already active channel.

In chained DMA operations, the TigerSHARC processor automatically sets up another DMA transfer when the entire contents of the current buffer have been transmitted or received:

- Transfer Control Block (TCB)

The chain pointer (field `CHPT` in `TCB` control register `DP`) is used to point to the next set of DMA parameters stored in internal memory. The chain target field (`CHTG`) in the `TCB` indicates the channel to be initiated.

- TCB chain loading

The DMA controller automatically reads the `TCB` from internal memory and loads the values into the channel `TCB` registers to set up the next DMA sequence at the end of the present one. This process is called `TCB` chain loading.

DMA Controller Operations

- DMA sequence

A DMA sequence is defined as the sum of the DMA transfers for a single channel, from the TCB registers initialization to when the count register decrements to zero.

Each TCB has a chaining enable bit (CHEN) in register DP. The channel is selected according to the value of the CHTG field in TCB_x register DP. The address in the CHPT field of the same register selects the TCB's internal memory address. (See “DP_x Register” on page 7-18.)

For external/internal memory transfers, the chaining between the internal and external channels must be coordinated. Both channels must have the chaining enable bit set.

Enabling and Disabling Chaining

DMA transfers are initiated by writing a quad-word from memory to the TCB_x register (EP channels require both TCB registers to be loaded). Chaining occurs if the chain enable bit is set, and the chain pointer is a valid address.

The CHPT field in TCB_x register DP can be loaded at any time during the DMA sequence. This allows a DMA channel to have chaining disabled (when the CHEN bit is set to 0 in TCB_x register DP) until some event occurs that loads the CHPT field with a target address, sets the CHEN bit, and defines the CHTG field. DMA chaining operations can be performed across I/O devices. While updating an active TCB, it must be paused by writing to the channel field in the DCNT register.

Interrupting Chaining

If the INT (interrupt enable) bit in TCB_x register DP is high, an interrupt occurs at the end of the current block transfer. Interrupts occur at the end of the chain sequence, instead of after each transfer, when the INT bit is set in the last chained TCB register.

Transfer Control Blocks and Chain Loading

During TCB chain loading, the DMA channel TCB registers are loaded with values retrieved from internal memory. The TCB is stored in four consecutive locations of an aligned quad-word.

The TCB for each DMA channel can be loaded under core control by executing a quad-word memory read instruction to the DMA TCB registers. The TCB can also be loaded through chaining or by an external direct write. For transfers between internal and external memories, two TCBs must be loaded.

TCB chain loading is requested like all other DMA operations. A TCB loading request is latched and held in the DMA controller until it becomes the highest priority request. If multiple chaining requests are present, the highest priority DMA channel is dealt with first. DMA channel request priorities are listed in Table 7-3 on page 7-38.

Setting Up and Starting the Chain

To set up and initiate a chain of DMA operations, the program should establish this sequence:

- Set up all TCBs in internal memory
- Write to the appropriate DMA control register

(TCB_x register DP):

- Set the CHEN (chaining enable) bit
- Define the CHTG (channel target)
- Define the CHPT (chaining pointer) fields, noting that the TCB's address is four times the value of this field

DMA Controller Operations

Chain Insertion

A high priority DMA operation or chain can be inserted into an active DMA chain. In order to do so the current channel transactions must be stalled by setting the corresponding pause bit in the `DCNT` register. A new DMA chain can be inserted into the current chain without affecting the current DMA transfer. The new chain is inserted by the TigerSHARC processor core by writing the `CHEN`, `CHTG`, and `CHPT` bits in the `TCBx` register `DP`. Once the `TCB` is updated, the `xpause` bit in register `DCNT` is reset and the data transfer continues from where it left off.

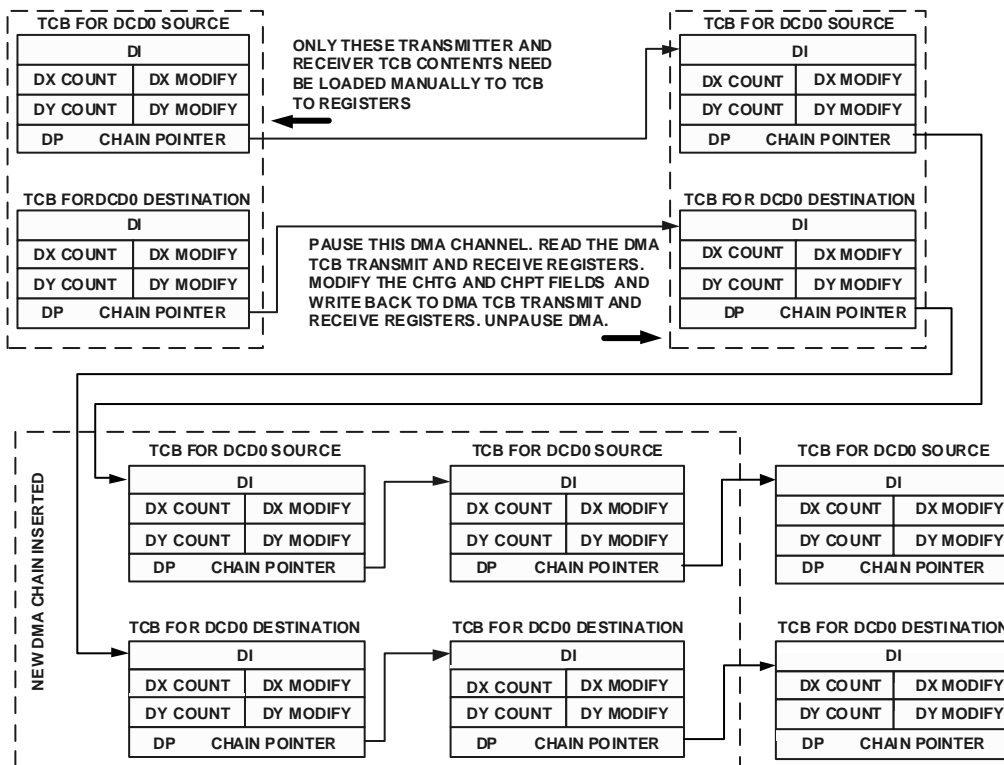


Figure 7-16. DMA Channel 0 Chain Insertion Example

Two-Dimensional DMA

This section describes the changes in functionality that occur when the TigerSHARC processor is placed in two-dimensional DMA mode. Two-dimensional DMA mode is enabled by setting the `ZDDMA` bit in the `TCB_DP` control register. This mode applies to external or internal addresses.

Two-Dimensional DMA Channel Organization

In two-dimensional mode, two-dimensional DMA array addressing can be performed on any DMA channel, receiver or transmitter.

The Index register (`TCB_DI`) is loaded with the first address in the data array and maintains the current address by adding the X modify after each transfer. The X Modify register (`TCB_DX_Modify`) contains the offset added to the current address to point to the next element in the X dimension (next column). The X Count register (`TCB_DX_Count`) and the `CIX` registers both initially contain the number of words in the X dimension. The `DCIX` register is used to reload `TCB_DX_Count` when it decrements to zero. `CX` is decremented after each transfer and contains the number of words left in the current row.

The Y Modify register (`TCB_DY_Modify`) contains the offset added to the current address to point to the next element in the Y dimension (first location in next row). When the X Count register reaches zero, this register is added to the current address on the following cycle, and the Y Count register is decremented. The value of `TCB_DY_Modify` should be the row distance. The modify value in the Y modify field in the `TCB` register `DY` is a 16-bit signed integer, allowing both increments and decrements. The 16 bits are sign-extended when used with the 32-bit index register.

The Y Count register (`TCB_DY_Count`) initially contains the number of words in the Y dimension (number of rows) and is decremented each time the X Count register reaches zero. When Y count reaches zero, the DMA

DMA Controller Operations

block transfer is complete. If chaining is enabled, the chain pointer field (CHPT) in the TCB DP register points to the start of a buffer in internal memory containing the next set of DMA parameters.

Two-Dimensional DMA Operation

A two-dimensional DMA transfer occurs in the following manner:

- **First Phase**
 - The current address stored in the TCB DI register is output and a DMA memory cycle is initiated.
 - In the same cycle, the X modify value stored in the TCB DX Modify register is added to the current address in the TCB DI register.
 - The TCB DX Count register is decremented.
 - If the decremented TCB DX Count is zero, the second phase is executed.
- **Second Phase**
 - The X count is restored into the TCB DX Count register from the DCIX register.
 - The Y increment value in the DMA register is added to the current address in the TCB DI register.
 - The TCB DY Count register is decremented.
 - If the Y count is zero, the DMA sequence is ended and the channel becomes inactive until the next pointer is written again.

A key feature of two-dimensional DMA sequence (or any DMA sequence) is the first DMA transfer begins before the address is modified. This means the DMA cannot be disabled by setting either the TCB DX Count or the TCB DY Count to zero. To disable two-dimensional DMA, the 2DDMA bit must be set to zero in the TCB DP Control register. To disable the DMA channel, the TY field should be reset in the TCB DP register.

When the X count becomes zero, but the Y count is nonzero, the X count must be reloaded with the original value. The DCIX register functions as the initial Count register while TCB DX Count decrements. The DCIX register holds the original count value that is restored in TCB DX Count register. The DCIX register is written automatically whenever the TCB DX Count register is loaded.

0	1	2	3	4	5	6	7
8	9	A	B	C	D	E	F
10	11	12	13	14	15	16	17
18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27
28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37
38	39	3A	3B	3C	3D	3E	3F

SUPPOSE WE WANT TO TRANSMIT AN 8X8 MATRIX OF DATA AS SHOWN ON THE RIGHT WITH QUAD OPERAND TRANSFERS. WHEN MAPPED TO MEMORY, THE VALUE 0 IS STORED IN MEMORY LOCATION 0X80100, 2 IS STORED IN 0X80102 AND SO ON. THE MEMORY LAYOUT WOULD EFFECTIVELY LOOK LIKE THAT SHOWN BELOW.

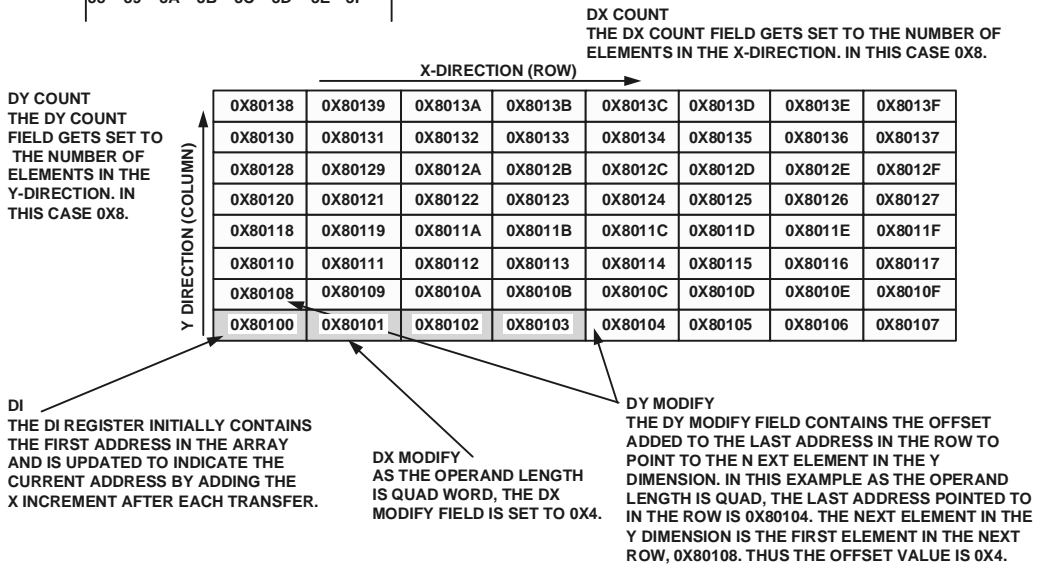



Figure 7-17. Two-Dimensional DMA of an 8x8 Array with Quad Operand Length

DMA Interrupts

An interrupt is generated by the DMA if the `INT` bit in the `TCBx` register `DP` is set (see “`DPx Register`” on page 7-18) and when the count register (the `X` count field in `CB` register `DX`) of an active DMA channel decrements to zero and the last transaction is completed. The count register(s) must decrement to zero as a result of the actual DMA transfers—merely writing zero to a count register does not generate the interrupt.

Each DMA channel `TCB` has its own interrupt. The DMA interrupts are latched in the `ILAT` register and are enabled in the `IMASK` register. For more information, see “DMA Interrupts” on page 4-5.

 Although the EP DMA channel access priority rotates, the interrupt priorities of all DMA channels are fixed.

As an alternative to interrupts, polling the `DSTAT` register can be used to determine when a single DMA sequence has completed. If chaining is enabled, however, polling the `DSTAT` should not be used because the next DMA sequence may be underway by the time the polled status is returned.

Starting and Stopping DMA Sequences

This section discusses starting, ending, suspending, and resuming DMA sequences.


Starting a DMA Sequence

A DMA sequence starts when one of the following occurs:

- The DMA channel is enabled, chaining is disabled, the DMA request bit is set, and a DMA request input transitions from high to low.
- The DMA channel is enabled, chaining is disabled, and the DMA request bit is cleared.

- Chaining is enabled, the DMA request bit is set, and a DMA request input transitions from high to low. Once the block transfer is completed, TCB chain loading of the channel TCB registers occurs and the DMA starts a new DMA sequence.
- Chaining is enabled and the DMA request bit is reset. Once the block transfer is completed, TCB chain loading of the channel TCB registers occurs and the DMA starts a new DMA sequence.

To start a new DMA sequence after the current one is finished, the program must write new parameters to the TCB registers.

 Writing an active TCB to an active TCB register causes a hardware error interrupt.

Ending a DMA Sequence

A DMA sequence ends when one of the following occurs:

- A channel is disabled by resetting the TY field in one of the TCB and DP registers.
- The count registers decrement to zero and chaining ends.

Suspending a DMA Sequence

A DMA sequence is suspended when the pause bit in one of the DCNT registers is set. (See “DCNTST Register” on page 7-28.)

Resuming a DMA Sequence

A DMA sequence is resumed when the pause bit in one of the DCNT registers is reset. (See “DCNTCL Register” on page 7-28.)

Whenever the DMA request goes low again, the DMA sequence continues from where it left off.

External Port DMA

Four DMA channels define internal/external memory transfers, where each channel has two `TCB DP` registers and where the `LEN` field in each one must be the same. These registers define the following.

- A channel is enabled if both `TY` fields in the `TCBx DP` registers are not set to 000.
- A channel's priority is high if the `PR` bit is set in either one of the `TCBx` and `DP` registers.
- DMA interrupt is enabled if the `INT` bit is set in either one of the `TCBx DP` registers.
- DMA request mode is enabled if the `DRQ` bit is set in either one of the `TCBx DP` registers.
- Internal and external memory transfer is set up in the `TY` field of the `TCBx` and `DP` registers. This allows efficient data transfers between the TigerSHARC processor's internal memory and external memory or devices.

The `CHTG` field must point to the operating channel in both `TCBs`' `DP` registers if chaining is enabled—for example, in `TCBs` of channel 1, the `CHTG` must be 1 in both `TCBs`. Each `TCB` register is loaded according to the respective `CHPT` fields.

Internal and External Address Generation

DMA transfers between the TigerSHARC processor internal memory and external memory require the DMA controller to generate addresses for both. The external address is generated according to the information in the external memory `TCB` register. (See “DMA Channel Control” on page 7-15.) The internal memory address is generated according to the information in the internal memory `TCB` register, where both `TCB` registers

belong to the same channel. If the $2DDMA$ bit in $TCBx DP$ register is set for one of the $TCBs$, the address is also updated with the modifier values in the $TCBx DY$ register. Internal-to-internal memory and external-to-external memory transfers are not supported, although external memory to and from external device transfers are supported.

External Port DMA Transfer Types

The TY field in the $TCBx DP$ register specifies the type of DMA transfer to perform. Each DMA channel has a transmitter and a receiver TCB register, where transmitters drive data and receivers get it.

External to Internal Memory

There are two ways to move data from external to internal memory:

- Program one of the four DMA channels to move a block from one memory to the other. In this case, the transmitter and receiver $TCBs$ should be programmed.

The configurations for transmitter and receiver $TCBs$ in this case are described in Table 7-4 on page 7-52 and Table 7-5 on page 7-53 respectively.

- Use one of the two AutoDMA register channels. In this case, the external device first programs the channel's TCB registers and then writes data to the target AutoDMA data register. Writing to this address invokes the corresponding DMA channel.

The configuration for the receiver TCB in this case is described in Table 7-6 on page 7-54.

External Port DMA

Table 7-4. Dual TCB Channel – External Memory TCB

Transmitter TCB Configuration		
Register	Field	Description
DI		External memory address
DX		Number of words to transfer and address modifier. The DX register also refers to X dimension data when the $2DDMA$ bit is set in the DP register.
DY		Number of Y dimension words to transfer and address modifier when the $2DDMA$ bit is set in the DP register; otherwise irrelevant.
DP	TY	External memory
DP	PR	Sets the channel priority 1 – Increases the channel priority – the cluster bus transaction is signaled as a \overline{DPA} (Priority Access) transaction
DP	$2DDMA$	Sets the two-dimensional mode 1 – Enables the two-dimensional mode – DY register becomes relevant DMA mode. (See “Two-Dimensional DMA” on page 7-45.)
DP	LEN	Can be word, double word or quad-word. Note the lowest internal bus utilization is achieved with quad-words.
DP	INT	Sets interrupt 1 – Interrupts the core after the complete block is transferred. (Enabled if set in either the source or destination $TCB \times DP$ register.)
DP	DRQ	Sets transfer mode 1 – Transfers each data item per \overline{DMARX} assertion. (Enabled if set in either the source or destination $TCB \times DP$ register.)
DP	CHEN	Sets chaining mode 1 – Enables chaining.
DP	CHTG	Defines the TCB register to be loaded; must be the same channel.
DP	CHPT	Chaining pointer – relevant when chaining is enabled.

Table 7-5. Dual TCB Channel – Internal Memory TCB

Receiver TCB Configuration		
Register	Field	Description
DI		Internal memory address
DX		Number of words to transfer and address modifier. The DX register also refers to X dimension data when the 2DDMA bit is set in the DP register.
DY		Number of Y dimension words to transfer and address modifier when the 2DDMA bit is set in the DP register; otherwise irrelevant.
DP	TY	Internal memory
DP	PR	Channel priority 1 – Increases the channel priority – internal bus DMA request priority is high.
DP	2DDMA	Sets the two-dimensional DMA mode 1 – Enables the two-dimensional DMA mode; the DY register becomes relevant (See “Two-Dimensional DMA” on page 7-45).
DP	LEN	Must be the same as the value specified in the LEN field of the transmitter’s TCB DP register.
DP	INT	Sets interrupt 1 – Interrupts the core once the complete block is transferred. (Enabled if set in either the source or destination TCBxDP register.)
DP	DRQ	Sets transfer mode 1 – Transfers each data item per $\overline{\text{DMARx}}$ assertion. (Enabled if set in either the source or destination TCBxDP register.)
DP	CHEN	Sets chaining mode; 1 – Enables chaining This setting must be identical to transmitter TCB.
DP	CHTG	Defines TCB register to be loaded; must be the same channel.
DP	CHPT	Chaining pointer – relevant when chaining is enabled.

External Port DMA

Table 7-6. AutoDMA Register Channel Operation
– Internal Memory TCB

Receiver TCB Configuration		
Register	Field	Description
DI		Internal memory address
DX		Number of words to transfer and address modifier. The DX register also refers to X dimension data when the 2DDMA bit is set in the DP register.
DY		Number of Y dimension words to transfer and address modifier when 2DDMA bit is set in the DP register; otherwise irrelevant.
DP	TY	Internal memory
DP	PR	Channel priority – must always be set to 1
DP	2DDMA	Sets the two-dimensional DMA mode 1 – Enables two-dimensional DMA mode; DY register becomes relevant. (See “Two-Dimensional DMA” on page 7-45.)
DP	LEN	Can be word, double word, or quad-word.
DP	INT	Sets interrupt 1 – Interrupts the core once the complete block is transferred.
DP	DRQ	Must always be set to 1.
DP	CHEN	Sets chaining mode 1 – Enables chaining
DP	CHTG	Defines TCB register to be loaded; must be the same channel.
DP	CHPT	Chaining pointer – relevant when chaining is enabled.

Internal to External Memory

Consider the case where the transfer direction is from internal to the external memory. The configurations for transmitter and receiver TCBs are described in Table 7-7 on page 7-55 and Table 7-8 on page 7-56 respectively.

Table 7-7. Internal Memory TCB

Transmitter TCB Configuration		
Register	Field	Description
DI		Internal memory address
DX		Number of words to transfer and address modifier. The DX register also refers to X dimension data when the 2DDMA bit is set in the DP register.
DY		Number of Y dimension words to transfer and address modifier when the 2DDMA bit is set in the DP register; otherwise irrelevant.
DP	TY	Internal memory
DP	PR	Channel priority 1 – Increases the channel priority – internal bus DMA request priority is high.
DP	2DDMA	Sets the two-dimensional DMA mode 1 – Enables two-dimensional DMA mode; the DY register becomes relevant. (See “Two-Dimensional DMA” on page 7-45.)
DP	LEN	Can be word, double word, or quad-word. Note the lowest internal bus utilization is achieved with quad-words.
DP	INT	Sets interrupt 1 – Interrupts the core once the complete block is transferred.
DP	DRQ	Sets transfer mode 1 – Transfers each data item per request.
DP	CHEN	Sets chaining mode 1 – Enables chaining.
DP	CHTG	Defines TCB register to be loaded; must be the same channel.
DP	CHPT	Chaining pointer – relevant when chaining is enabled.

External I/O Device to External Memory (Flyby)

Consider the case where the transfer direction is from an external I/O device to the external memory. There is no internal memory access. The configurations for transmitter and receiver TCBs are described in Table 7-9 on page 7-57 and Table 7-10 on page 7-58 respectively.

External Port DMA

Table 7-8. External Memory TCB

Receiver TCB Configuration		
Register	Field	Description
DI		External memory address
DX		Number of words to transfer and address modifier. The DX register also refers to X dimension data when the 2DDMA bit is set in the DP register.
DY		Number of Y dimension words to transfer and address modifier when the 2DDMA bit is set in the DP register; otherwise irrelevant.
DP	TY	External memory
DP	PR	Channel priority 1 – Increases the channel priority – cluster bus transaction is signaled as a Priority Access transaction.
DP	2DDMA	Sets the two-dimensional DMA mode 1 – Enables two-dimensional DMA mode; the DY register becomes relevant. (See “Two-Dimensional DMA” on page 7-45.)
DP	LEN	Must be the same as the value specified in the LEN field of the transmitter’s TCB DP register.
DP	INT	Sets interrupt 1 – Interrupts the core once the complete block is transferred. Irrelevant when the INT bit is set in the transmitter’s TCB DP register.
DP	DRQ	Sets transfer mode 1 – Transfers each data item per request. Irrelevant when the DRQ bit is set in the transmitter’s TCB DP register.
DP	CHEN	Sets chaining mode 1 – Enables chaining The setting must be identical to transmitter TCB.
DP	CHTG	Defines TCB register to be loaded; must be the same channel.
DP	CHPT	Chaining pointer – relevant when chaining is enabled.

For more information, see “Flyby Transactions” on page 5-34.

Table 7-9. I/O Device TCB
 – External I/O Device to External Memory (Flyby)

Transmitter TCB Configuration		
Register	Field	Description
DI		Irrelevant
DX		Number of words to transfer and address modifier; modify must be 0.
DY		Irrelevant
DP	TY	External I/O device (Flyby)
DP	PR	Irrelevant
DP	2DDMA	Must be cleared.
DP	LEN	Can be either word or double word.
DP	INT	Sets interrupt 1 – Interrupts the core once the complete block is transferred.
DP	DRQ	Sets transfer mode 1 – Transfers each data item per request.
DP	CHEN	Sets chaining 1 – Enables chaining.
DP	CHTG	Defines TCB register to be loaded; must be channel 0.
DP	CHPT	Chaining pointer – relevant when chaining is enabled.

External Memory to External I/O Device (Flyby)

Consider the case where the transfer direction is from external memory to an external I/O device. There are no internal memory accesses and the data held in the OFIFO is irrelevant. The configurations for transmitter and receiver TCBs are described in Table 7-11 on page 7-59 and Table 7-12 on page 7-60 respectively.

For more information, see “Flyby Transactions” on page 5-34.

External Port DMA

Table 7-10. External Memory TCB
– External I/O Device to External Memory (Flyby)

Receiver TCB Configuration		
Register	Field	Description
DI		External memory address
DX		Number of words to transfer, increment is irrelevant. The DX register also refers to X dimension data when the 2DDMA bit is set in the DP register.
DY		Number of Y dimension words to transfer and address modifier when the 2DDMA bit is set in the DP register; otherwise irrelevant.
DP	TY	External memory
DP	PR	Channel priority 1 – Increases the channel priority – cluster bus transaction is signaled as a Priority Access transaction
DP	2DDMA	Sets the two-dimensional DMA mode 1 – Enables two-dimensional DMA mode; the DY register becomes relevant. (See “Two-Dimensional DMA” on page 7-45.)
DP	LEN	Must be the same as the value specified in the LEN field of the transmitter’s TCB DP register.
DP	INT	Sets interrupt 1 – Interrupts the core once the complete block is transferred. Irrelevant when INT bit is set in the transmitter’s TCB DP register.
DP	DRQ	Sets transfer mode 1 – Transfers each data item per request. Irrelevant when DRQ bit is set in transmitter’s TCB DP register.
DP	CHEN	Sets chaining mode 1 – Enables chaining. Irrelevant when the CHEN bit is set in the transmitter’s TCB DP register.
DP	CHTG	Defines TCB register to be loaded; must be channel 0.
DP	CHPT	Chaining pointer – relevant when chaining is enabled.

Table 7-11. I/O Device TCB
 – External Memory to External I/O Device (Flyby)

Transmitter TCB Configuration		
Register	Field	Description
DI		External memory address
DX		Number of data elements to transfer, increment is irrelevant. The DX register also refers to X dimension data when the 2DDMA bit is set in the DP register.
DY		Number of Y dimension words to transfer and address modifier when the 2DDMA bit is set in the DP register; otherwise irrelevant.
DP	TY	External memory
DP	PR	Sets channel priority 1 – Increases the channel priority – cluster bus transaction is signaled as a PA (Priority Access) transaction.
DP	2DDMA	Sets the two-dimensional mode 1 – Enables two-dimensional DMA mode; the DY register becomes relevant. (See “Two-Dimensional DMA” on page 7-45.)
DP	LEN	Can be either word or double word.
DP	INT	Sets interrupt 1 – Interrupts the core once the complete block is transferred
DP	DRQ	Sets transfer mode 1 – Transfers each data item per request.
DP	CHEN	Sets chaining mode 1 – Enables chaining.
DP	CHTG	Defines TCB register to be loaded; must be channel 0.
DP	CHPT	Chaining pointer – relevant when chaining is enabled.

DMA Semaphores

Since DMA channels are a system resource, a device that wants to use a DMA channel must determine whether one is currently available. It is recommended this be done through software, using an agreed upon location

External Port DMA

Table 7-12. External Memory TCB
– External Memory to External I/O Device (Flyby)

Receiver TCB Configuration		
Register	Field	Description
DI		Irrelevant
DX		Number of words to transfer and address modifier.
DY		Irrelevant
DP	TY	External I/O device (Flyby)
DP	PR	Irrelevant
DP	2DDMA	Must be cleared.
DP	LEN	Must be the same value as the LEN field in the transmitter's TCB DP register.
DP	INT	Sets interrupt 1 – Interrupts the core once the complete block is transferred. Irrelevant if the INT bit is set in the transmitter's TCB DP register.
DP	DRQ	Sets transfer mode 1 – Transfers each data item per request. Irrelevant if the DRQ bit is set in the transmitter's TCB DP register.
DP	CHEN	Sets chaining 1 – Enables chaining. Irrelevant if the CHEN bit is set in the transmitter's TCB DP register.
DP	CHTG	Defines TCB register to be loaded; must be channel 0.
DP	CHPT	Chaining pointer – relevant when chaining is enabled.

in each processor's memory to specify which channels are currently available. This memory location should only be modified using a read-modify-write operation.

Handshake Mode

Each of the four EP DMA channels can be driven by a request control— $\overline{\text{DMAR0}}$, $\overline{\text{DMAR1}}$, $\overline{\text{DMAR2}}$ and $\overline{\text{DMAR3}}$.

An access is requested when the external device pulls $\overline{\text{DMARx}}$ low. The falling edge is detected by the TigerSHARC processor and synchronized to the processor's I/O clock. In order to be recognized in a particular cycle, the $\overline{\text{DMARx}}$ low transition must meet the setup time specified in the data sheet, otherwise it may take effect in the following cycle. The DMA channel x (same channel as $\overline{\text{DMARx}}$) issues the transactions according to the programming in its TCBs. If the source is external, or the transaction is Flyby, the transaction is issued to the cluster bus via OFIFO. If the source is internal, the transaction is issued on the internal bus, and the data read from the source is written to the cluster bus via OFIFO.

The external device does not have to wait for the actual bus transaction before making another request. The requests are stored in a working counter maintained internally by the TigerSHARC processor. The counter holds a maximum of 15 requests, so the external device can make up to 15 requests before the first one has been serviced.



More than 15 requests without a DMA transfer is likely to cause unpredictable results.

The associated $\overline{\text{DMARx}}$ signal is ignored if a DMA channel is disabled.

Flyby Mode

In Flyby mode, the TigerSHARC processor operates as an independent DMA controller. Flyby mode transfers are similar to standard DMA transfers; although it is important that the data width match the external I/O device's width.

The $\overline{\text{DMAR0}}$ register retains the same functionality in this mode. The TigerSHARC processor outputs: addresses, $\overline{\text{MSx}}$ memory selects, $\overline{\text{FLYBY}}$ and $\overline{\text{RD}}/\overline{\text{WR}}$ strobes, and responds to $\overline{\text{ACK}}$. The external memory access behaves

External Port DMA

exactly as if the TigerSHARC processor core had requested it—the FIFOs latch or drive data if the transfer is to or from internal memory. Finally, the TCB for the DMA channel must be preloaded to generate the external memory addresses and word count.

The TY field in the TCB0 DP register (which is external I/O device) and the DRQ field in the same register (where DMA request is enabled) allow the DMA to be driven by the DMA request lines, and the I/O device to be controlled by the $\overline{\text{FLYBY}}$ and $\overline{\text{TOEN}}$ signals. This way, the Bus Interface Unit can interface external memory and external I/O devices in a single cycle. For more information, see “Flyby Transactions” on page 5-34.

Several TigerSHARC processors in a multiprocessing cluster can share a $\overline{\text{FLYBY}}$ signal. The $\overline{\text{FLYBY}}$ register is driven by the current TigerSHARC processor bus master only with a DMA external transfer when the TY field in its associated TCB DP register is defined as external I/O.

When the I/O device wants to be read, it asserts the $\overline{\text{DMAR0}}$ signal—each assertion (falling edge) of the $\overline{\text{DMAR0}}$ signal represents one transaction. The TigerSHARC processor responds by executing a write transaction to memory and asserts $\overline{\text{TOEN}}$ to indicate when to drive the data, and asserts $\overline{\text{FLYBY}}$ to select the appropriate data to drive. When the I/O device requires data from memory, it also asserts $\overline{\text{DMAR0}}$. The TigerSHARC processor responds by executing a read transaction from memory and asserts $\overline{\text{FLYBY}}$ to the I/O device—in this case, the $\overline{\text{TOEN}}$ signal is inactive.

The I/O device can execute up to 15 $\overline{\text{DMARx}}$ signals before the TigerSHARC processor executes the first transaction in response to it.

For asynchronous and pipeline transfers see “Flyby Transactions” on page 5-34.

When requesting an access, the external device pulls $\overline{\text{DMARx}}$ low—the falling edge is detected by the TigerSHARC processor and synchronized to the processor’s clock. In order to be recognized in a particular cycle, the $\overline{\text{DMARx}}$ low transition must meet the setup time specified in the data sheet, otherwise it could take effect in the following cycle.

Link Ports DMA

Four DMA channels define link ports to internal memory, external memory, or other link ports transfers. Another four channels define internal memory, external memory, or other link ports to link ports transfers.

Link Ports DMA Transfer Types

The `TY` field in the `TCBx DP` register specifies the type of DMA transfer to be used. Receiving link DMA channels have a receiver `TCB` register, and transmitting link DMA channels have a transmitter `TCB` register.

Link Port to Internal/External Memory

In order to move data from link ports to internal or external memory, the receiver `TCB` must be programmed. The DMA initiates a transfer by making a request of the internal bus. As a result, data is transferred from the link port to internal or external memory (OFIFO). The `TCB` configuration is described in Table 7-13.

Internal/External Memory to Link

Consider the case where the transfer direction is from internal or external memory to link port.

- **Internal memory to Link**
The DMA initiates a transfer by requesting data from internal memory. At this stage, data is loaded to the link port when it is available on the internal data bus.
- **External memory to Link**
The DMA initiates an internal transfer to fill up the OFIFO buffer, and the external port starts an output to the external device (one or more times depending on the packing mode). Link address is put into IFIFO with read data, whereupon the IFIFO requests an

Link Ports DMA

Table 7-13. Link to Internal/External Memory TCB

Receiver TCB Configuration		
Register	Field	Description
DI		Internal/external memory address
DX		Number of words to transfer and address modifier. The <i>DX</i> register also refers to X dimension data when the <i>2DDMA</i> bit is set in the <i>DP</i> register.
DY		Number of Y dimension words to transfer and address modifier when the <i>2DDMA</i> bit is set in the <i>DP</i> register; otherwise irrelevant.
DP	TY	Internal/external memory
DP	PR	Channel priority 1.....Increases the channel priority – internal/cluster bus DMA request priority is high.
DP	2DDMA	Sets the two-dimensional DMA mode 1 – Enables two-dimensional DMA mode; the <i>DY</i> register becomes relevant. (See “Two-Dimensional DMA” on page 7-45.)
DP	LEN	Always set to quad-word.
DP	INT	Sets interrupt 1 – Interrupts the core once the complete block is transferred.
DP	DRQ	Don't care (set in hardware).
DP	CHEN	Sets chaining 1 – Enables chaining.
DP	CHTG	Defines the TCB register to be loaded; may be any link channel.
DP	CHPT	Chaining pointer – relevant if chaining is enabled.

internal bus access to link port. Once the bus is granted, it drives the data on the data bus. The transmitter TCB configuration is described in Table 7-14.

Receiving Link Port to Link Port

In order to move data from one link port to another link port, the receiver TCB should be programmed as a link TCB. See Table 7-15.

Table 7-14. Internal/External Memory to Link TCB

Transmitter TCB Configuration		
Register	Field	Description
DI		Internal/external memory address
DX		Number of words to transfer and address modifier. The DX register also refers to X dimension data when the 2DDMA bit is set in the DP register.
DY		Number of Y dimension words to transfer and address modifier when the 2DDMA bit is set in the DP register; otherwise irrelevant.
DP	TY	Internal/external memory
DP	PR	Channel priority 1 – Increases the channel priority – internal/cluster bus DMA request priority is high.
DP	2DDMA	Sets the two-dimensional DMA mode 1 – Enables two-dimensional DMA mode; DY register becomes relevant. (See “Two-Dimensional DMA” on page 7-45.)
DP	LEN	Always set to quad-word.
DP	INT	Sets interrupt 1 – Interrupts the core once the complete block is transferred.
DP	DRQ	Don't care (set in hardware)
DP	CHEN	Sets chaining 1 – Enables chaining.
DP	CHTG	Defines TCB register to be loaded; may be any link channel.
DP	CHPT	Chaining pointer – relevant when chaining is enabled.

The DMA initiates a transfer by requesting the internal bus, and data is transferred from the link port that requested the DMA services to the receiver link port.

DMA Throughput

Table 7-15. Link to Link TCB

Receiver TCB Configuration		
Register	Field	Description
DI		Link port address <ul style="list-style-type: none">• 0x1804A0 for link #0• 0x1804A8 for link #1• 0x1804B0 for link #2• 0x1804B8 for link #3
DX		Number of words to transfer and address modifier – the address modifier must be set to 0.
DY		Irrelevant
DP	TY	I/O link ports or DMA data registers
DP	PR	Channel priority 1 – Increases the channel priority—internal bus DMA request priority is high.
DP	2DDMA	Cleared.
DP	LEN	Always set to quad-word.
DP	INT	Sets interrupt 1 – Interrupts the core once the complete block is transferred.
DP	DRQ	Don't care (set in hardware).
DP	CHEN	Sets chaining 1 – Enables chaining.
DP	CHTG	Defines TCB register to be loaded; may be any link channel.
DP	CHPT	Chaining pointer – relevant if chaining is enabled.


DMA Throughput

This section discusses overall DMA throughput when several DMA channels are trying to access internal or external memory at the same time.

Internal Memory DMA

The DMA channels and I/O devices arbitrate for access to the TigerSHARC processor's internal memory and buses. The DMA controller determines, on a cycle-by-cycle basis, which channel is allowed access to the three internal buses or OFIFO, and consequently which channel reads or writes to internal memory. The priority of the DMA channels is discussed in “DMA Channel Prioritization” on page 7-36.

There is no overall throughput loss in switching between channels. Any combination of link ports and external port transfers has the same maximum internal transfer rate, which is one DMA transfer per cycle.

 The internal memory clock rate and bus widths are at least twice the external port clock rate and bus width. Thus the bandwidth of each internal bus is at least four times greater than the external port bandwidth.

In the case of link ports, assuming all the links operate at half the core frequency, the maximum link port bandwidth is 1/16 of an internal bus bandwidth.

External Memory DMA

When the DMA transfer is between the TigerSHARC processor internal memory and external memory, the external memory may have one or more wait states. External memory wait states, however, do not reduce the overall internal DMA transfer rate if other channels have data available to transfer. The TigerSHARC processor internal data buses are not held up by an incomplete external transfer.

When data is to be transferred from internal to external memory, the internal memory data is placed in the external port's OFIFO, completing the OFIFO write request. The external memory access then begins independently. For external-to-internal DMA, the internal bus DMA request

DMA Operation on Boot

is not made until the external memory data is at the output of the IFIFO. In both cases, the external DMA address is maintained in the address FIFO until the data transfer is completed.

In Flyby mode, DMA transfers occur between an external device and external memory. This means that no internal resources of the TigerSHARC processor, other than the assigned DMA channel, are utilized and internal DMA throughput is not affected.

DMA Operation on Boot

DMA operation on booting is described in “DMA Operation on Boot” on page 10-32.

8 LINK PORTS

The TigerSHARC processor link port is an optional communication channel. It is intended to be used for point-to-point communications between TigerSHARC processors in a system, but can be used to interface to any other device that is designed to work in the same protocol.

There are four link ports on the TigerSHARC processor. Each link port is an 8-bit data bidirectional half-duplex port with three additional control pins.

Data is driven and latched on both the rising and falling edge of the link clock. Each link port can transfer data at the rate of one byte per $CCLK$ cycle, up to 250 MBytes/second. Automatic token switching changes the direction of transfer.

Transfer control at the core occurs through interrupts or polling. Transfer control at the DMA occurs through the dedicated DMA channels for transmit and receive.

DMA chaining is supported and all link ports can be used for booting.

TigerSHARC processor link ports are not compatible with SHARC processor link ports.

Link Architecture

The link port has two parts—transmitter and receiver. Each has a shift register and buffer, as shown in Figure 8-1 on page 8-2. The LBUFTx and LBUFRx registers are memory-mapped Uregs. The shift registers are not accessible by software. All are 128-bit registers.

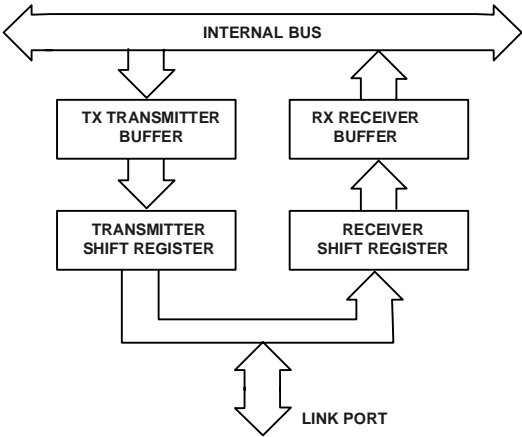


Figure 8-1. Link Port Architecture

Link I/O Pins

Table 8-1 describes the I/O pins related to the link ports. There is a set of link pins for each link port. The 'x' in the signal name indicates the link port—0, 1, 2, or 3.

Table 8-1. Link I/O Pins

Signal	Type	Description
LxCLKIN	Input	Clock when link port x receives. Acknowledge when link port x transmits.
LxCLKOUT	Output	Clock when link port x transmits. Acknowledge when link port x receives.
LxDAT7-0	I/O	Data input and output for link port x.
LxDIR	Output	Indicates whether link port x is transmitting or receiving. 1 link port x is the output 0 link port x is the input

Link ports should be connected as shown in Figure 8-2. The minimal system only uses the LxCLKIN, LxCLKOUT, and LxDAT pins. The LxCLKOUT pin of each TigerSHARC processor is connected to the LxCLKIN of the other. The LxDAT bus is connected between the two TigerSHARC processors.

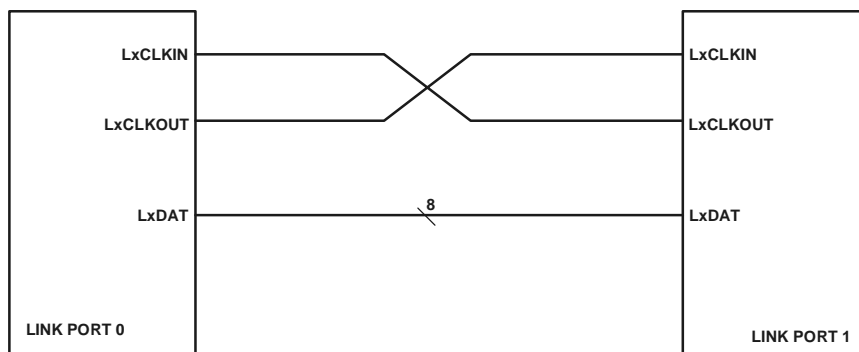


Figure 8-2. Minimal Link Configuration – No Buffering

An optional system is shown in Figure 8-3. This configuration uses the LxDIR pin in addition to the LxCLKIN, LxCLKOUT, and LxDAT pins. This configuration might be used when differential low swing buffers are used for long twisted-wire pairs. The figure shows a set of bidirectional buffers

Link Architecture

between the LxDAT pins that are controlled by LxDIR. The LxCLK buffers can be regular buffers. The delays on all LxCLK and LxDAT signals must be exactly matched. Buffers must be disabled by $\overline{\text{RESET}}$ (the output LxDIR is three-stated by the TigerSHARC processor.)

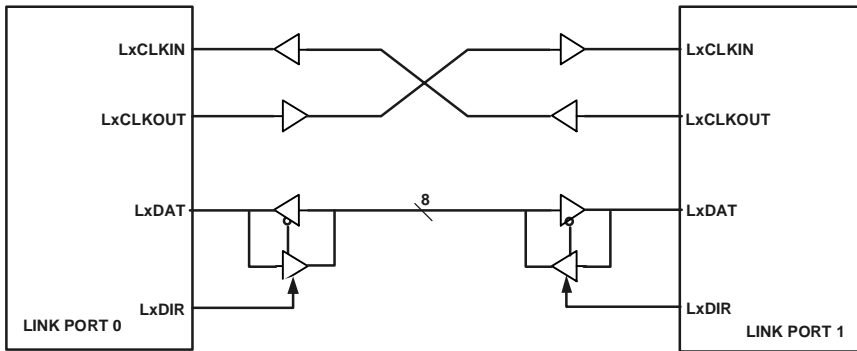


Figure 8-3. Buffered Link Configuration Using LxDIR

Transmitting and Receiving Data

Data is transferred by writing to the LBUFTx register and reading from the LBUFRx register. Core driven transfer is performed by the core writing quad-word data to the LBUFTx register, and the core reading quad-word data from the LBUFRx register. DMA driven transfer is performed through two DMA channels per link port, one for transmit and one for receive. The link port DMA requires a Transfer Control Block (TCB) for either transmit or receive, and only quad-word transfers are allowed. DMA chaining is supported.

All the data written to the LBUFTx register is first copied to the shift register when it is empty, and then transmitted. After the data is copied to the shift register, new data can be written to the LBUFTx register.

Table 8-2. Link Port Register – Group 0x25

Register Number	Register Quads	Links 0-3 Registers	Address	Remarks
Reg 0x0 - 3	LBUFTX0	Link # 0 Transmit register	0x1804A0 - 3	
Reg 0x4 - 7	LBUFRX0	Link # 0 Receive register	0x1804A4 - 7	Read only
Reg 0x8 - B	LBUFTX1	Link # 1 Transmit register	0x1804A8 - B	
Reg 0xC - F	LBUFRX1	Link # 1 Receive register	0x1804AC - F	Read only
Reg 0x10 - 13	LBUFTX2	Link # 2 Transmit register	0x1804B0 - 3	
Reg 0x14 - 17	LBUFRX2	Link # 2 Receive register	0x1804B4 - 7	Read only
Reg 0x18 - 1B	LBUFTX3	Link # 3 Transmit register	0x1804B8 - B	
Reg 0x1C - 1F	LBUFRX3	Link # 3 Receive register	0x1804BC - F	Read only

The link transmitter tries to transmit all the data that is written to the transmit shift register. The link receiver enables data movement only when the receive shift register is empty. At this point, the received data is shifted into the receive shift register. After the whole quad-word is received, the receiver waits until the LBUFR_x register is free and copies the data from the shift register to the LBUFR_x register. When the shift register is free again, it can enable receiving data again.

DMA


Each link port is associated with two DMA channels. One channel is used for transmitting data while the other is used for receiving data. The two DMA channels can interface with either internal or external memory.

Link Architecture

Table 8-3. Link Port Register – Group 0x27

Register Number	Register Quads	Reserved for Control Registers and Links 4-5 Registers	Direct Memory Address	Remarks
Reg 0x0	LCTL0	Link # 0 control register	0x1804E0	Reset value 0x400 (LREN=1)
Reg 0x1	LCTL1	Link # 1 control register	0x1804E1	Reset value 0x400 (LREN=1)
Reg 0x2	LCTL2	Link # 2 control register	0x1804E2	Reset value 0x400 (LREN=1)
Reg 0x3	LCTL3	Link # 3 control register	0x1804E3	Reset value 0x400 (LREN=1)
Reg 0x10	LSTAT0	Link # 0 status register	0x1804F0	Read only
Reg 0x11	LSTAT1	Link # 1 status register	0x1804F1	Read only
Reg 0x12	LSTAT2	Link # 2 status register	0x1804F2	Read only
Reg 0x13	LSTAT3	Link # 3 status register	0x1804F3	Read only
Reg 0x18	LSTATC0	Link # 0 status clear register	0x1804F8	Read only
Reg 0x19	LSTATC1	Link # 1 status clear register	0x1804F9	Read only
Reg 0x1A	LSTATC2	Link # 2 status clear register	0x1804FA	Read only
Reg 0x1B	LSTATC3	Link # 3 status clear register	0x1804FB	Read only

The link port issues a DMA request to the transmit DMA channel when the LBUFTx register is empty and the DMA channel is enabled. The link port issues a DMA request to the receive link DMA channel when it receives a data quad-word—that is, when the LBUFRx register is full and the DMA channel is enabled.

 If the transmit buffer is empty (for transmit channel) or if the receive buffer is full (for receive channel), a DMA request is issued as soon as a link DMA channel (transmit or receive) becomes active.

The receive DMA can also be used for pass-through transfers by writing to the transmit register of another channel. The DMA also takes care to transfer data only when the transmit register of the target link is free.

Interrupts

If the receive link DMA channel has not been initialized, the link port interrupt is issued by the link receiver when a quad-word received by the link is waiting in the `LBUFRx` register. If the DMA channel associated with this link receive channel then becomes active after the interrupt is issued, the interrupt goes off and a DMA request is issued. Since the link interrupt is a level-triggered interrupt, it is ignored.

The interrupt can be used for two purposes:


- As an error indication when the system designer intends to manage the data by using the DMA.
- For control indication between different TigerSHARC processors. The link interrupt routine can read the link receiver buffer, analyze it, and behave accordingly. For more information, see “Link Port DMA Control” on page 7-32.

Reset and Boot

The link ports can also be used for boot. After reset, all link ports are initialized to receive mode. The link receive DMA channel is initialized to receive 256 words and write them to internal memory block 0, address 0. For more information, see “Boot Link to Internal Memory” on page 10-36.

Link Port Communication Protocol

The DMA channel of any port not being used for boot must be cleared and reinitialized. If the DMA channel is not deactivated, the link interrupt mechanism does not work. The link may be reinitialized as well.

-  If both links that are connected to each other start transmitting at the same time, unexpected results may occur (up to protocol deadlock). The system designer must initialize the system in such a way that one of the links transmits first.

Link Port Communication Protocol

The link port communicates through an 8-bit data bus, using three control signals. There is a set of link pins for each link port. In the link pin names, the “x” stands for the link port—0, 1, 2, or 3. The control signal pins are:

- **LxCLKIN** – The **LxCLKIN** pin is an input that functions as a clock when the link receives, and as an acknowledge when the link transmits.
- **LxCLKOUT** – The **LxCLKOUT** pin is an output that functions as a clock when the link transmits, and as an acknowledge when the link receives.
- **LxDAT7-0** – Data bus input and output
The data is transmitted or received one byte every **LxCLK** edge. Transfers are of quad-words. The transfer begins from byte 0.
- **LxDIR** – Link direction indication
The **LxDIR** pin is an output that indicates if the link is transmitting or receiving. It is used when the links are buffered.

The link port protocol includes a Transfer Acknowledge and an optional Connectivity Check for each quad-word transferred. Errors resulting from these handshakes are described in “Error Detection Mechanisms” on page 8-17.

During the Connectivity Check the transmitter detects whether the receiver is connected. The receiver pulses its $LxCLKOUT$ low when it begins to receive data. The transmitter checks for a low pulse on its $LxCLKIN$ during transmission of its current quad-word.

The Transfer Acknowledge is a handshake for each quad-word transferred. The receiver drives $LxCLKOUT$ high to indicate that it is ready for the next quad-word or drives it low to indicate a stall. The transmitter checks for its $LxCLKIN$ to be high before transmitting the next quad-word.

The minimum granularity of the transfer is a quad-word transferred in eight cycles (16 bytes, on both clock edges). The transfer is initiated by the transmitter. The transmitter can initiate a transfer only if the receiver has set its $LxCLKOUT$ (which is the transmitter $LxCLKIN$) high, meaning it is in receive mode and has a free buffer.

An example for transmit start is shown in Figure 8-4. Note $LxCLKIN$ and $LxCLKOUT$ are shown from the transmitter's point of view. This is also the case in the figures shown on successive pages, unless otherwise stated.

Link Port Communication Protocol

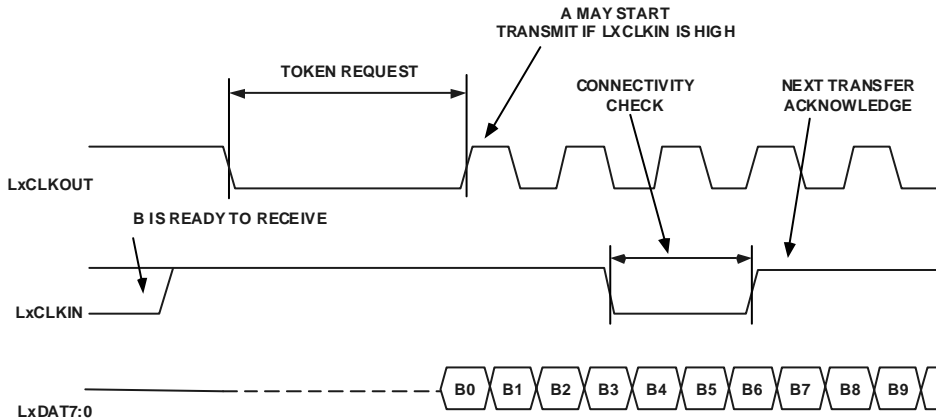


Figure 8-4. Link A to Link B Begins Transmission

1. A token request is issued to the receiver when the transmitter drives its LxCLKOUT low.
2. During the token request the transmitter waits for six cycles to verify that LxCLKIN is high, indicating that it is ready to receive (Transmit Acknowledge), then enables transmission start.
3. The transmitter drives the quad-word data on to the LxDAT7-0 lines, one byte at a time, least significant byte first, at the rising and falling edges of its LxCLKOUT.
4. One cycle after the transmission starts, the receiver drives the transmitter's LxCLKIN low for a connectivity check. If LxCLKIN remains low, indicating a receiver stall, another quad-word cannot be transmitted until the receive buffer is free. When the buffer is empty, the receiver sets LxCLKIN high (Transmit Acknowledge), to be ready for another token request.

The $LxCLKOUT$ is used as the synchronization clock, driven by the transmitter. Data is latched in the receive buffer on the falling and rising edges of $LxCLKOUT$.

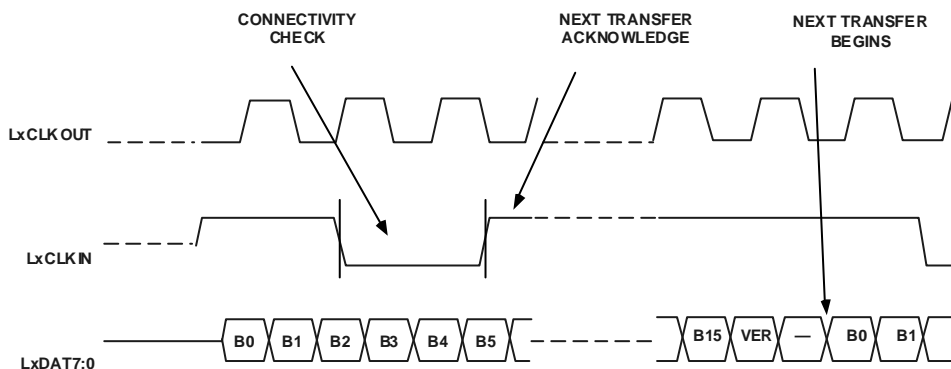


Figure 8-5. Quad-word Completion and a New Quad-word Beginning When $VERE$ Bit is Set

The completion of the transfer and the beginning of a new transfer can be done according to one of two scenarios, according to the $VERE$ bit in the $LCTLx$ register (enable verification). If the $VERE$ bit is set, the checksum byte is sent after the last byte in quad-word, on the rising edge. In this case, the falling edge data is meaningless and the checksum byte is sampled at the $LxCLKIN$ rising edge. If the $VERE$ bit is cleared, the first byte of the next quad-word follows the last byte of the current word.

The $LxCLKIN$, driven by the receiver to the transmitter, is always used as a “wait” indicator and sometimes used as a connectivity check to ensure that the receiver is getting the current transmission.

When used as a wait indicator, $LxCLKIN$ is driven low. If $LxCLKIN$ stays low, the transmitter can complete the current quad-word, but it cannot begin a new quad-word transmission. If there is more data to be transmitted, the transmitter sets the $LxCLKOUT$ to low and waits until $LxCLKIN$ is driven high by the receiver.

Link Port Communication Protocol

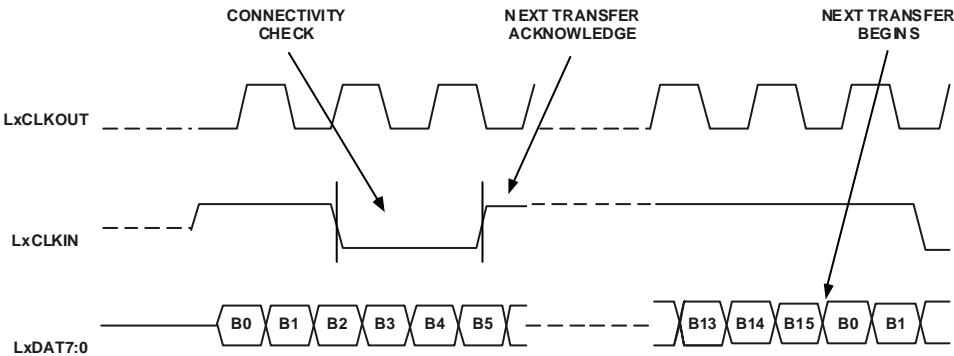


Figure 8-6. Quad-word Completion and a New Quad-word Beginning When VERE Bit is Cleared

If LxCLKIN is set high before the 12th edge, there back-to-back transmissions occur.

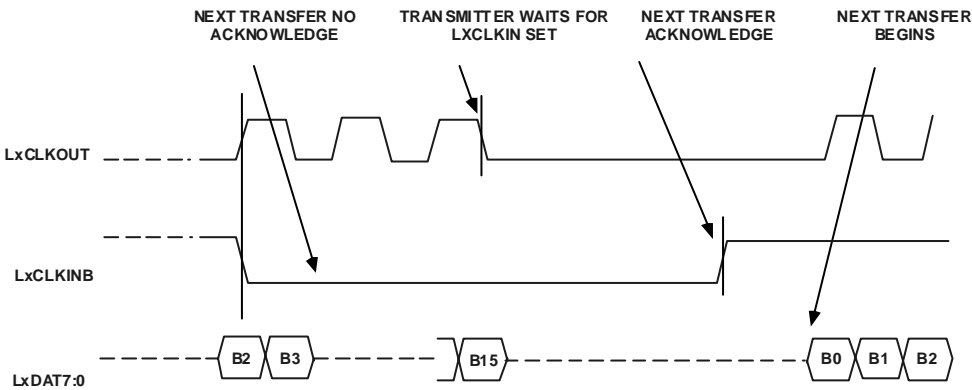


Figure 8-7. Transfer Waits Because Receiver Buffer is Full

A token switch is the exchange between the transmitter (master) and the receiver (slave) roles. Only the transmitter can enable the token switch. The transmitter enables a token switch in one of these two cases:

- No more data to transmit
- If the `PSIZE` bit in `LCTLx` is cleared (64 quads packet size), after transmitting 64 quad-words

Figure 8-8 shows an example of a token switch. In this case, the `LxCLK` names are shown from the first transmitter's point of view (TigerSHARC processor A). The transmitter indicates transmission completion by setting the `LxCLKOUT` high at the end of the last quad-word transmission.

When TigerSHARC processor B (the receiver) senses that its input `LxCLKIN` (which is connected to `LxCLKOUT` of TigerSHARC processor A) is high for the period between one and two `LxCLK` cycles, it can request a token switch by setting `LxCLKOUT` output low. The `LxCLK` cycle period is defined by the link clock divisor and `CCLK`.

The token is switched if `LxCLKOUT` remains high long enough, thus indicating the former transmitter (A in this example) did not regret the token switch and can receive data. (Refer to the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*.)

The link has an output `LxDIR` that indicates data direction. When `LxDIR` is high, the link data pins direction is output. The `LxDIR` pin is cleared when the transmitter enables a token switch. The `LxDIR` pin is set after the token switch, when the data is transmitted—the high value does not overlap. This control can be used by any type of bidirectional buffers on the data lines.

The former transmitter can regret the token switch if it gets more data to transmit after a pause, and if the former receiver has not yet requested a token switch.

Link Port Communication Protocol

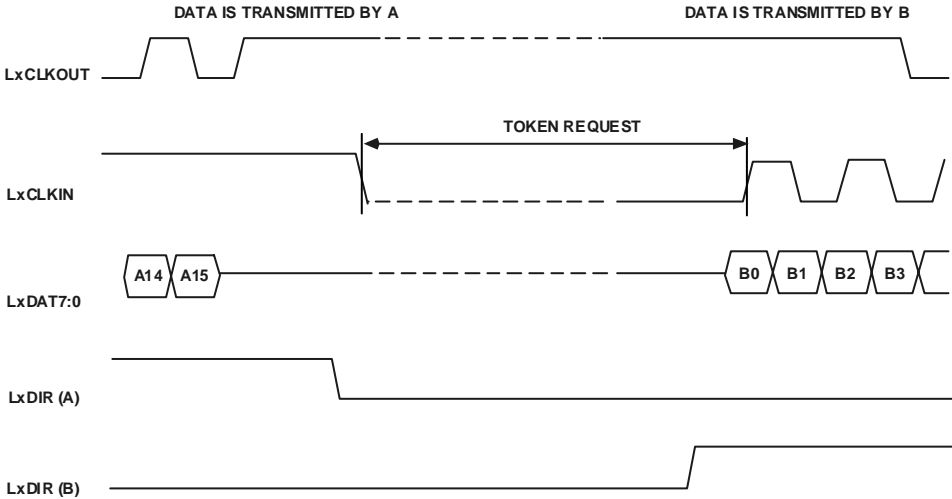


Figure 8-8. Token Switch from A to B

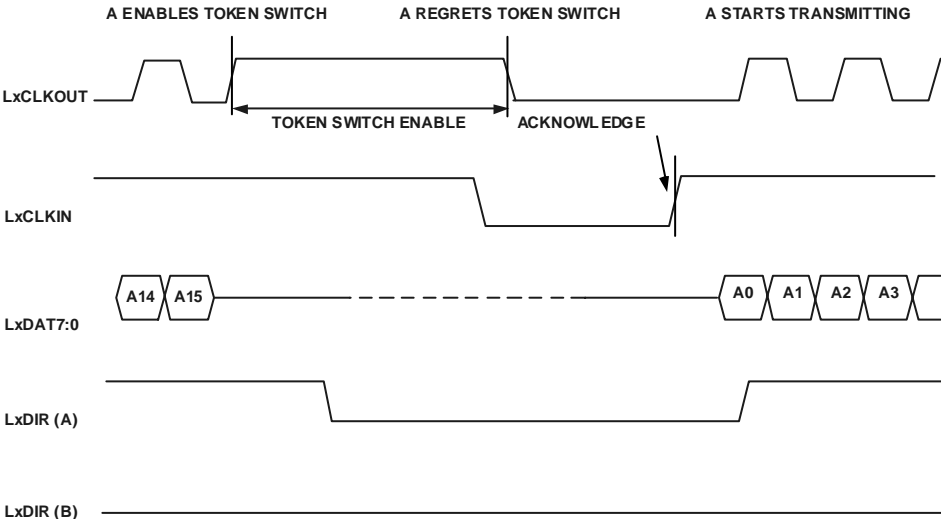


Figure 8-9. Transmitter Regrets a Token Switch

The former transmitter must keep the $LxCLKOUT$ high long enough to guarantee the token switch takes place. (See the ADSP-TS101 TigerSHARC Embedded Processor Data Sheet.) The receiver must request the token switch when it senses its $LxCLKIN$ is high. If the receiver does not request the token switch in time and the transmitter has more data to send, the transmitter indicates that it regrets the token switch by setting $LxCLKOUT$ low (token request), which cancels the token switch.

An example of a token switch regret is shown in Figure 8-9. In this example:

- After the transmitter has set $LxCLKOUT$ high, the receiver identifies this as token switch enable and requests the token switch.
- The receiver's request is too late to guarantee the token switch and the transmitter signals a regret by setting $LxCLKOUT$ low. The receiver identifies this and goes back to being a receiver. The receiver indicates that it is ready to receive more data by acknowledging. The transmitter starts transmitting again.
- The $LxDIR$ pin of the transmitter changes to low when it enables the token switch. The transmitter sets its $LxDIR$ back to high only when the transmitter starts to transmit data again. The receiver does not set its $LxDIR$ output high because it never gets to the point of transmitting data.

Transmission Delays

The links should be able to function even if there are delays between the source and destination. The delays of the different link signals ($LxCLK$ and $LxDAT$) are matched to tolerances specified in the data sheet.

Transmission Delays

The data sampled at the receiver uses the input $LxCLK$, so the delay has no effect. Complications may arise, however, when token switches are involved. The maximum delay can be derived from information provided in the data sheet.

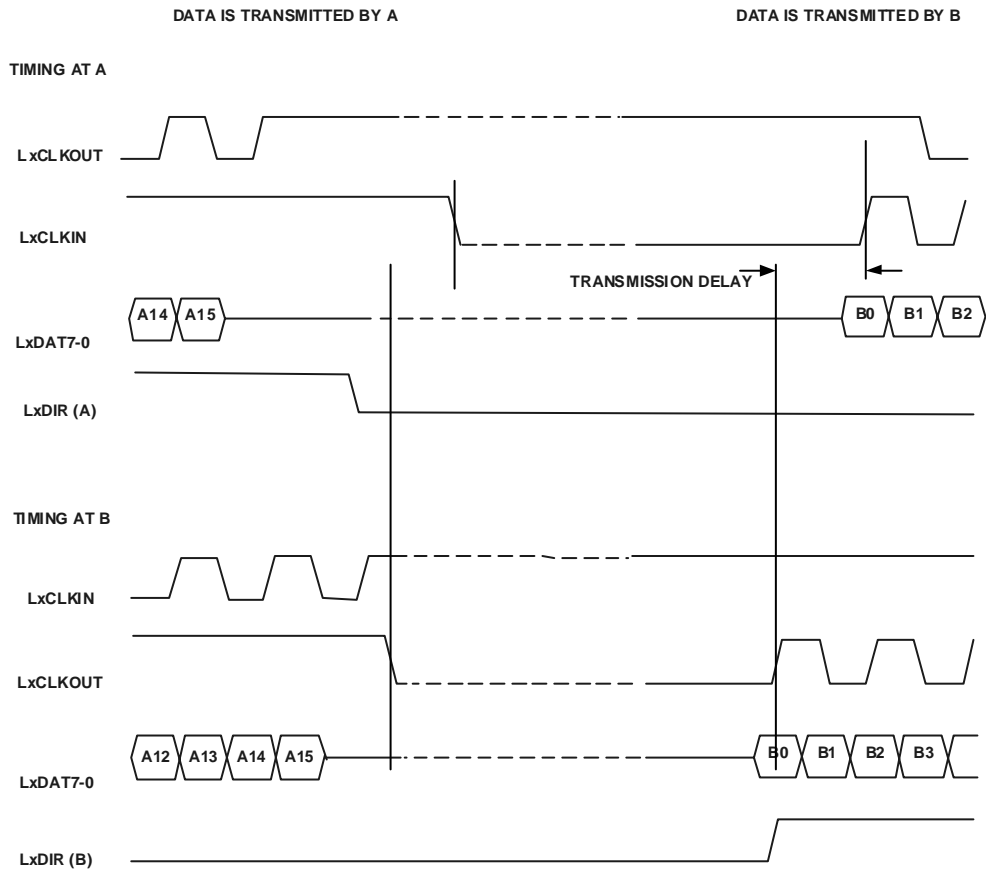


Figure 8-10. Token Switch From A to B With Delay

Figure 8-10 shows the effect of the delay between two TigerSHARC processors. The transmitter (TigerSHARC processor A) sets $LxCLKOUT$ high to indicate a token switch enable. The receiver (TigerSHARC processor B) senses this after a delay. The receiver then sets its $LxCLKOUT$ (which is $LxCLKIN$ of the transmitter) low and waits to see if the transmitter regrets the token switch. The token switch takes place if the transmitter has not regretted the token switch. Refer to the data sheet for specific timing data.

Error Detection Mechanisms

The link ports support on-the-fly error checks. When the link detects an error condition, it operates in the following sequence.

- The TER or RER field in $LSTATx$ is set according to the error type.
- If the $TTOE$ (Transmit Time Out check Enable), $CERE$ (Connectivity Error check Enable), or $RTOE$ (Receive Time Out check Enable) bits are set in the $LCTLx$ registers, a hardware error interrupt is set. (See “Hardware Error Operations” on page 4-8.)
- If the $VERE$ (checksum Verification Error Enable) is set in the $LCTLx$ registers, the RER field in $LSTATx$ is set according to this error type, and a hardware error interrupt is set. (See “Hardware Error Operations” on page 4-8.)
- The link no longer continues to transmit/receive until the error status is read (from the $LSTATCx$).

After reading the status from the destructive address, it should be reinitialized to start working.

Transmitter Error Detection

Two kind of transmitter errors are detected.

- **Connectivity check**

The transmitter checks the `LxCLKIN` pin on every quad transmission. The pin acts as the acknowledge signal. If the acknowledge signal is not deasserted for a minimum time period during the transmission, an error has occurred—the `TER0` bit is set in the `LSTATx` register and a hardware interrupt is issued. Refer to the data sheet for specific timing data.

- **Time out**

If the transmitter's acknowledge signal (`LxCLKIN`) is deasserted for a time period of 2048 clock cycles, and the transmitter has data to transmit, an error has occurred—the `TER1` bit is set in the `LSTATx` register, and a hardware interrupt is issued.

Receiver Error Detection

Two kinds of receiver errors are detected.

- **Checksum**

The transmitter creates and sends for a special verification byte on every transmitted quad-word (if it is enabled). This byte is a checksum calculation of all data bytes that have been transmitted. The byte is sent at the end of every 16-byte (quad-word) transmission. The receiver compares the transmitted verification byte to the local verification byte created during data reception. If the two bytes differ, the `RER0` bit is set in the `LSTATx` register and a hardware interrupt is issued.

The checksum algorithm is:

$$\text{Checksum} = B0 + B1 + \dots + B15 + C0 + C1 + \dots + C13$$

where B0, B1, and so on are byte1, byte2 ...

C0 is the Carry_out from B0 + B1,

C1 is the Carry_out from B0 + B1 + B2 + C0,
and so on

C15 (the Carry_out from B0 + B1 + ... + B15 + C0 + C1 + ... + C13) is not added to the checksum

- **Time out**

If `LxCLKIN` is sampled low for a period of 256 cycles, a timeout error has occurred. The `RER1` bit is set in the `LSTATx` register and a hardware interrupt is issued. If less than 16 bytes are received and `LxCLKIN` remains high for more than four `LxCLK` cycles, the receiver does not detect a timeout error. The next quad-word received overwrites the data in the receive buffer.

Control Register (LCTLx)

The control register programs every link port and contains control bits unique to each one. There are four control registers—one for each link port. The register is read/write.

Control Register (LCTLx)

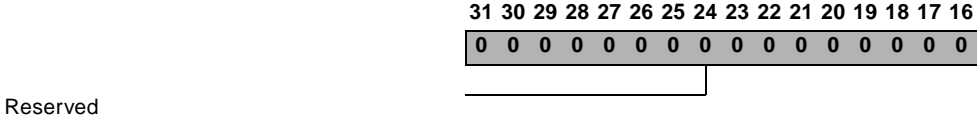


Figure 8-11. LCTLx (Upper) Register Bit Descriptions

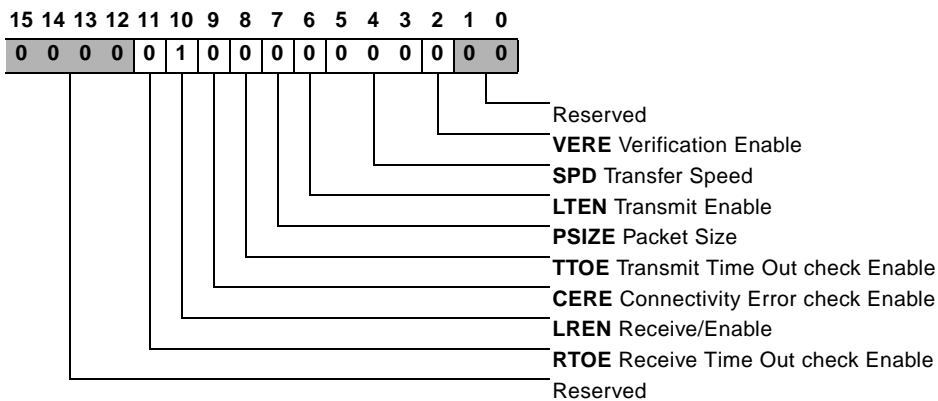


Figure 8-12. LCTLx (Lower) Register Bit Descriptions

Control Register (LCTLx)

Table 8-4. LCTLx Register Bit Descriptions

Bits	Name	Description
1-0		Reserved
2	VERE	Verification Enable Enables checksum verification for received data and creation of checksum on data transmission. Clear after reset.
5-3	SPD	Transfer Speed Defines the communication rate in which bytes are transmitted/received ^{1,2} 000 – CCLK/8 001 – CCLK/4 010 – CCLK/3 011 – CCLK/2 1XX – Reserved This field is clear after reset.
6	LTEN	Transmit Enable When cleared, any ongoing transmit procedure inside the block ceases immediately, and data is three-stated. Any data that has been written to the transmitter register before clearing LTEN is cleared, and when setting LTEN again, the transmit register will be empty. Upon reset, this bit is cleared.
7	PSIZE	Packet Size When a packet is completed, the transmitter enables token switch. 0-64 quads in one packet 1 – Indefinite number of bytes on one packet Upon reset, this bit is cleared.
8	TTOE	Transmit Time Out check Enable Enables hardware error interrupt on time out check for transmitted data. When set, a hardware error interrupt is issued. Upon reset, this bit is cleared.
9	CERE	Connectivity Error check Enable Enables hardware error interrupt on connectivity check error to transmit link port. When set, a hardware error interrupt is issued. Upon reset this bit is cleared.
10	LREN	Receive/Enable When disabled, any ongoing receive procedure inside the block is ceased immediately and any received data is cleared. When set, the receive register is empty. Upon reset, this bit is set.

Table 8-4. LCTLx Register Bit Descriptions (Cont'd)

Bits	Name	Description
11	RTOE	Receive Time Out check Enable Enables hardware error interrupt on time out check for received data. When set, a hardware error interrupt is issued. Upon reset this bit is cleared.
31–12	Reserved	

- 1 Both transmitter and receiver have to work at the same clock speed, although the clock doesn't need to be fully synchronous.
- 2 Refer to the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet* for timing specifications and frequency limitations.

After reset:

- The register is initialized to 0x400—receive enable.
- Transmit is disabled; packet size is 64 quad-words.
- Clock division is by 8.
- DIR pin is disabled.
- Verification is disabled and all error checks are disabled.

In normal operation, the programmer configures all Link Control register features just once after reset.

Status Register (LSTATx)

The Status register holds error and status information for every link port. There are four status registers, one for each link port. The status registers are read-only—note however, that a read from a clear address clears the error fields RER and TER.

Status Register (LSTATx)

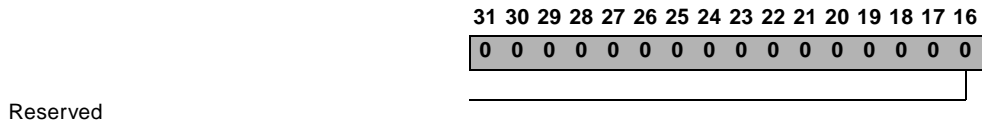


Figure 8-13. LSTATx (Upper) Register Bit Descriptions

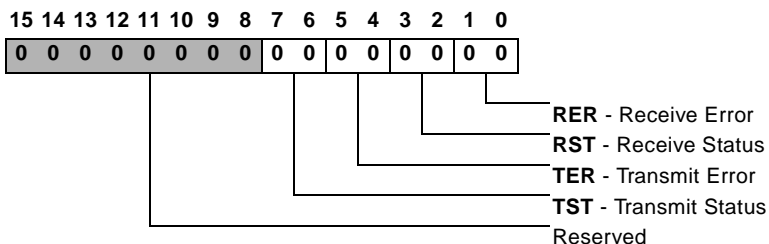


Figure 8-14. LSTATx (Lower) Register Bit Descriptions

Table 8-5. LSTATx Register Bit Descriptions

Bits	Name	Description
1-0	RER	Receive Error 00 – No error 01 – Verification byte error 10 – Receive time out 11 – Both errors
3-2	RST	Receive Status 00 – Receiver empty 01 – Receiver shift register partly full, LBUFRx buffer free 10 – Receiver LBUFRx buffer full, shift register empty 11 – Receiver full
5-4	TER	Transmit Error 00 – No error 01 – Connectivity error 10 – Transmit time out 11 – Both errors Note any error causes hardware error interrupt. Reading the Status register clears the error and resets the state machine for a new transfer.
7-6	TST	Transmit Status 00 – Transmitter empty 01 – Transmitter partly full 11 – Transmitter full
31-8	Reserved	

Status Register (LSTATx)

9 DEBUG FUNCTIONALITY

The functionality described in this chapter is useful for both software debugging and services usually found in operating system (OS) kernels. These features are implemented in the TigerSHARC processor hardware to allow debuggers and operating systems to build the more complex aspects of their features. The functionality is grouped into levels. These levels build upon each other to allow greater control and visibility of the TigerSHARC processor for the debugger or kernel.

The debugging approach is based on the idea of protected execution (user mode) and unprotected execution (supervisor mode). In user mode, some of the system resources can be made accessible. However, when those resources must be managed by an OS kernel, they are restricted. When user code tries to access a protected resource, an exception occurs and the access is aborted.

The debugger can control the TigerSHARC processor in one of two ways:

- using a debug monitor (running in supervisor mode)
- using an In-circuit emulator (ICE)

In supervisor mode, some system resources are allocated to the monitor. (For example, memory, communication channel-to-host, interrupt, flag bits, and so on.)

The ICE, on the other hand, uses a reserved communication channel (the JTAG test access port) to control the TigerSHARC processor. This way the ICE can be non-intrusive to the system, as well as control systems that cannot communicate with a host system.

Operating Modes

Since many of the functions needed by the debug monitor and the ICE overlap, most of the debug functions in the TigerSHARC processor are shared between the two methods. There are some features that need to be optionally restricted by the ICE so that the debugger can be used in supervisor mode.

Debug capabilities are available either through the ICE or through the supervisor mode. As already mentioned, the ICE mode is accessed through the JTAG port. The supervisor mode is entered via $\overline{\text{Reset}}$, an exception, or a specific software watchpoint. When the TigerSHARC processor is in supervisor or emulation mode, it has complete access to all of the TigerSHARC processor's resources.

Table 9-1 details debug concepts and capabilities.

Operating Modes

The TigerSHARC processor operates in one of three modes—emulator, supervisor, and user. In user and supervisor modes, *all* instructions are executed normally. In user mode, however, the register access is limited. In emulation mode, *none* of the control flow instructions (excluding `If True, RTI (np);`) may be used—control flow instructions may cause problems and it is illegal to use them in emulation. Operating modes are discussed in “Operation Modes” on page 3-2.

Debug Resources

Special Instructions

The TigerSHARC processor supports special instructions (traps) which are used to aid system debugging. These instructions are required to implement both software breakpoints (in debuggers) and operating system calls (for OS kernels).

Table 9-1. Debug Concepts

Debug Tools	Debug Feature
Watchpoints	Allows a user to specify particular watchpoint address ranges and conditions that halt the processor when those conditions are met. The user can then examine the state of the processor, restore that state, and continue instruction execution. Software breakpoints and single step are implemented using watchpoint as well.
Multiprocessing Capability	Allows a user to single step and set breakpoints in a multiprocessor system.
Trace History	Allows a trace of the program counter to be recreated. An on-chip trace buffer (FIFO) stores the last eight branches (<i>all</i> discontinuous values of the program counter) taken by the program sequencer, allowing the recent program path to be recreated.
Cycle Count	Provides the basic functionality for all code profiling functions.
Performance Monitoring	Allows internal resources to be monitored unobtrusively. The debugger can specify which internal resource(s) should be monitored.
Access to Protected Registers	Provides access to protected registers. Protected registers are only accessible in supervisor mode or ICE. They cannot be accessed by user mode and are protected from accidental modifications that could cause system damage.
Watchpoint Counters	Allows the user to search for the nth occurrence of an event before halting.
Exception	Aborts execution of the next and the following piped instructions. For more information, see “Exceptions” on page 4-23.

Watchpoints

Address watchpoints allow the user to examine the state of the processor whenever a set of user-defined memory access conditions occurs.

There are three watchpoint sets that can operate in parallel. At each watchpoint, the user can define conditions for bus access cycles and the operation the TigerSHARC processor should execute when these occur.

Debug Resources

Programming – Control and Address Pointer Registers

Each watchpoint has two address registers— WP_{iL} and WP_{iH} . See “Watchpoint Address Pointers – WP_{0L} , WP_{1L} , WP_{2L} , WP_{0H} , WP_{1H} and WP_{2H} ” on page 2-29 for more information. When the watchpoint is programmed for a single address, only the low address register (WP_{iL}) is used. If the watchpoint is searching for an address range, the WP_{iL} holds the low address, and the WP_{iH} holds the high address. The address range must be contained within a single internal memory block or within the external memory space. The only exception is watchpoint 0 when it is programmed to search for the sequencer instruction fetch. In this case, the range is unlimited. When the address range is external, the DMA external accesses are not monitored.

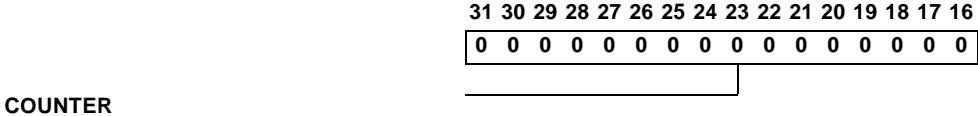


Figure 9-1. WP_{xCTL} (Upper) Register Bit Descriptions

It is important to set the Address Pointer registers first (before the $WPiCTL$ register) which define the actual watchpoint operation. The address registers of a watchpoint cannot be changed when the watchpoint is active.

A watchpoint operation is defined by its Control register— $WPiCTL$ (where i is 0, 1, or 2 for different watchpoints). $WPiCTL$ registers are configured as follows.

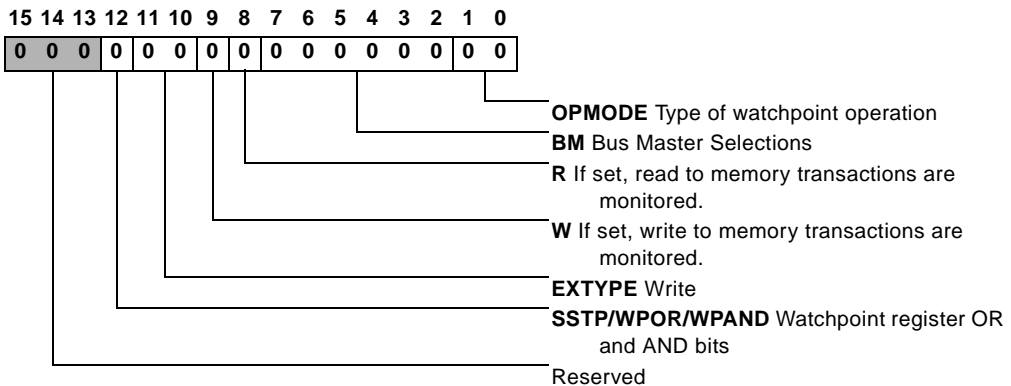


Figure 9-2. $WPxCTL$ (Lower) Register Bit Descriptions

Table 9-2. WPiCTL Register Bit Descriptions

Bits	Bit	Description
1–0	OPMODE	<p>Operation Mode</p> <p>Determines the type of watchpoint operation.</p> <p>00 – Watchpoint disabled and not in use.</p> <p>01 – Watchpoint is used to compare a single address (breakpoint). In this case, the low Watchpoint Address register defines the target address.</p> <p>10 – Watchpoint register is used to define an address range. The two Watchpoint registers define the range of search.</p> <p>11 – Determines address ranges beyond the usual range. The search is for transactions inside the memory block, but outside of the range (addresses lower than the lowest address defined by the Watchpoint register, or higher than the highest address defined by the Watchpoint register).</p>
7–2	BM	<p>Bus Master Selections</p> <p>Any bus master or any combination of bus masters can be selected to trigger the event. Every master has one bit, when if set, the watchpoint monitors the transactions initiated by this module. The transactions of the core (sequencer, J-IALU, and K-IALU) are referred to only when initiating instruction executions have completed, and only if these are not aborted. For more information, see “Watchpoint Control – WP0CTL, WP1CTL and WP2CTL” on page 2-25.</p>
8	R	<p>Read</p> <p>If set, read to memory transactions are monitored.</p>
9	W	<p>Write</p> <p>If set, write to memory transactions are monitored.</p>
11–10	EXTYPE	<p>Exception Type</p> <p>Determines the procedure following count experiment.</p> <p>00 – No exception. This option is used to get an indication of the output of the $\overline{\text{EMU}}$ pin.</p> <p>01 – Regular software exception is performed.</p> <p>10 – Emulation trap is executed.</p>

Table 9-2. WPiCTL Register Bit Descriptions (Cont'd)

Bits	Bit	Description
12	SSTP/WPOR/WPAND ¹	<p>Watchpoint register OR and AND bits</p> <p>In WP1CTL and WP2CTL Bit12 of the control register is WPOR and WPAND. When one of these is set, the three watchpoint sets work together. If the WPOR bit is set in WP1CTL, the counter is decremented every time one of the watchpoints finds a match. If the WPAND bit is set in WP2CTL, the counter is decremented every time all the watchpoints find a match in the same cycle.² Note this bit is meaningless in WPOCTL—in WPOCTL, this bit overrides the setup of the watchpoint.</p>
15–13		Reserved
31–16	COUNTER	<p>Watchpoint Counter Initialization</p> <p>Defines how many times the event should be triggered before it causes an exception or emulation trap. It can be set between 1 and 64K-1. Note zero may not be programmed.</p>

¹ When WPOR and WPAND are set, all three watchpoints must be initialized and enabled.

² WPOR can be used to detect accesses that cross memory block boundaries. Split the range in the memory block and each subrange can be detected by a different watchpoint set.

Watchpoint Operation

The following information must be programmed into the Watchpoint registers before a search can begin:

- Address or address range
- Master initiating the transaction
- Count

Debug Resources

When the control register is set to a non-idle operation, the count field is set to the initial count. After the count field is reset, the watchpoint hardware monitors the internal buses for:

- Transaction addresses matching what is specified in the watchpoint address registers.
- The identity of the master that had initiated a transaction when it matches one of those indicated by the control register.

Whenever there is a match, the counter is decremented.

After the counter reaches zero, an exception is initiated—either a supervisor exception or an emulator trap according to Bit11–10 in the `EXtype` field in `WPiCTL`. At this point, the Status register indicates ‘watchpoint count expired’. To begin another search, the user must rewrite the control register.

Watchpoint Status (WPISTAT)

The `WPISTAT` register indicates the current status of the search. It holds the current count of the search and the mode the watchpoint is executing. When the counter expires, the watchpoint status Bits17–16 change to 11.

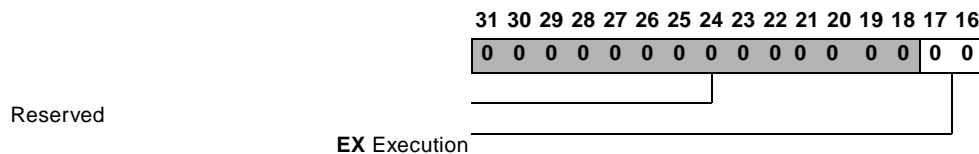


Figure 9-3. WPxSTAT (Upper) Register Bit Descriptions

Table 9-3. WPISTAT Register Bit Descriptions

Bits	Name	Description
15–0	VALUE	Watchpoint Value Determines the current value of the watchpoint counter.
17–16	EX	Execution Bit Watchpoint is executing 00 – Watchpoint disabled 01 – Searching for match 11 – Watchpoint count expired
31–18	Reserved	Reserved

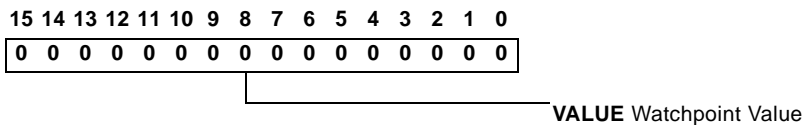


Figure 9-4. WPxSTAT (Lower) Register Bit Descriptions

Instruction Address Trace Buffer (TBUF)

Since the program counter is not accessible in real time off chip when executing instructions from internal memory, a trace buffer has been provided to assist code debugging.

Performance Monitors

The trace buffer is an eight register list that stores a history of the last four to eight branches taken by the program sequencer, allowing the user to fully recreate the path the program took for the last eight branches.

The branches are written into the eight Trace Buffer registers in a circular manner. The first is written into `TRCB0`, the next into `TRCB1`, and so on until `TRCB7`. The ninth branch is written back into `TRCB0`, and so on. In order to ascertain the last register, the user must refer to the Trace Buffer pointer register, which points to the last written entry.

The trace buffer always holds the PC of the branch instruction line. In case of a computed branch, it also stores the target PC of the branch on the next entry. To distinguish between the jump PC and the target PC, the `TRCBMASK` should be used. Every bit in the `TRCBMASK` is associated with the corresponding `TRCB i` register. If it is cleared, the `TRCB i` holds the PC of the jump instruction line. If set, the `TRCB i` holds the target PC of the jump.

Performance Monitors

The performance monitors are targeted to optimize the functioning of a working application. The monitors provide indications of how many times an event occurs during a specific run. These monitors are constructed of three registers—cycle counter, performance monitor mask, and performance monitor counter. The cycle counter and performance monitor should be initialized to zero by the user before a specific run. The ratio between the total count and the performance monitor counter gives the required indication.

When the performance monitor counter expires, it issues an exception.

Cycle Counter (CCNT1-0)

CCNT1-0 counts cycles while the program is executing. *All* cycles are counted. The cycle counter is a 64-bit counter that increments with every cycle, including executions in both user and supervisor modes. All hold, wait state, aborted instructions, and other delay cycles are also counted.

When the TigerSHARC processor enters emulator mode, the cycle counter halts, since it is irrelevant.

Performance Monitor Mask – PRFM

The Performance Monitor Mask is a 32-bit register that defines the events that are monitored by the performance counters. Every bit in the Performance Monitor refers to an event that reflects the application performance (for example, the number of stall cycles in the system). See “Performance Monitor Mask – PRFM” on page 2-23.

When only one bit in the PRFM register is set, the feature indicated by this bit is counted while the TigerSHARC processor is running in user and supervisor mode. The count is halted when the TigerSHARC processor is in emulation mode. For certain events (for example, compute block instructions or bus transactions), the counter may compute more than one transaction/instruction every cycle, according to the event occurrence.

When more than one bit is set in the mask, the count is according to the SUM bit (Bit 31). If the SUM bit is clear, and more than one bit in the Performance Mask is set, CCNT1-0 counts every cycle where one of the events occurs. The counter, however, does not sum the events but ORs them. If the SUM bit is set, the events are summed. In this case, on a cycle where the compute block (X or Y) executes two instructions, the counter is incremented by two, and on bus transactions (Bits14–12) the counter could be incremented by three.

JTAG Functionality

Where the `SUM` bit is set, there are restrictions on the different events that can be combined. The set bits must be a subset of one of the following groups.

- Bits11–0 – granted or non granted bus requests
- Bits20–12 – transactions per bus
- Bits29–21 – instructions

The Performance Monitor Mask is disabled by clearing all bits in the mask.

Performance Monitor Counter

The Performance Monitor counts the events according to the programmed mask. It counts the events while the TigerSHARC processor is in user or supervisor mode.

JTAG Functionality

The TigerSHARC processor supports the IEEE Standard 1149.1 Test Access Port.

Pins

Table 9-4 describes those TigerSHARC processor pins used by JTAG emulation hardware. Underlined or overbarred pin names indicate negative true signals.

Table 9-4. JTAG and Emulation I/O Pins

Signal	Type	Description
$\overline{\text{TRST}}$	Input Asynchronous	Test Reset (JTAG) Resets the test state machine. The $\overline{\text{TRST}}$ signal must be asserted after power up to ensure proper JTAG operation. The $\overline{\text{TRST}}$ signal has a 100 K Ω internal pull-up resistor.
TCK	Input	Test Clock (JTAG) Provides an asynchronous clock for JTAG boundary scan.
TDI	Input	Test Data Input (JTAG) A serial data input of the boundary scan path. This signal has a 100 K Ω internal pull-up resistor.
TDO	Output	Test Data Output (JTAG) A serial data output of the boundary scan path.
TMS	Input	Test Mode Select (JTAG) Controls the test state machine. This signal has a 100 K Ω internal pull-up resistor.
$\overline{\text{EMU}}$	Output	Emulation Connected to the ADSP-TS101 DSP EZ-ICE target board connector only.

JTAG Functionality

As an extension to the JTAG standard, the TMS and $\overline{\text{EMU}}$ pins have these additional debugging functions:

- The TMS pin (when T_{EME} bit in the EMUCTL register is set) functions as an emulation exception pin. An emulation exception is issued on every rising edge of the TMS signal. This function is in addition to TMS functionality in JTAG.
- The $\overline{\text{EMU}}$ output is an open drain signal (for wired OR) driven only if the EMUOE bit in the EMUCTL register is set. The $\overline{\text{EMU}}$ pin gives the following indications:
 - On watchpoint event no exception (EX_{type} field in WPiCTL) is 00.
 - The $\overline{\text{EMU}}$ is pulled down for 20 CCLK cycles.
Whenever the TigerSHARC processor is in emulation mode and ready for a new instruction line driven into the EMUIR register, the $\overline{\text{EMU}}$ pin is driven low until such an instruction is inserted. For more information, see “Emulation Mode” on page 3-3.

JTAG Instruction Register

The JTAG Instruction register is five bits long and allows the Test Access Port (TAP) controller to select any of the scannable data registers. The LSB is the first to be shifted out through the TDO. The encoding of this Instruction register follows the TigerSHARC processor implementation.

Data Registers

The Data registers are selected via the Instruction register. Once a particular Data register's value is written into the Instruction register, and the TAP state is changed to SHIFT-DR, the particular data going in and out of the TigerSHARC processor depends on the definition of the Data register selected. See the IEEE 1149.1 specification for more details.

Table 9-5. JTAG Port Access Instruction Decode

If IR4–0 Value is...	Corresponding Instruction is...	To...
00000	EXTEST	Execute EXTEST
10000	Sample/Preload	Execute sample and preload
01000	EMUIR	Scan in EMUIR register
10100	EMUDAT	Scan EMUDAT register
01110	IDCODE	Scan ID code register
11110	SAMPLEPC	Sample PC from which the TigerSHARC processor is running
10001	EMUTRAP	Execute an emulation trap
01001	WPOCS	Scan out WPO status, and scan in WPO control
11001	WP0L	Scan WPO low
00101	WPOH	Scan WPO high
10101	WP1CS	Scan out WP1 status, and scan in WP1 control
01101	WP1L	Scan WP1 low
11101	WP1H	Scan WP1 high
00011	WP2CS	Scan out WP2 status, and scan in WP2 control
10011	WP2L	Scan WP2 low
01011	WP2H	Scan WP2 high
00100	EMUCTL	Scan EMUCTL register
01100	EMUSTAT	Scan EMUSTAT register
11111	BYPASS	Bypass

When registers are scanned out of the device, the LSB is the first bit to go out of the TigerSHARC processor.

JTAG Functionality

The data registers are:

- **Bypass**

A IEEE 1149.1 required register, this is a one (1) bit register.

- **Boundary**

Registers used by multiple JTAG instructions. All four of the JTAG instructions that use the boundary are required by the IEEE 1149.1 specification.

- **EMUIR**

The JTAG instruction loads 48 bits, of which the 32 LSBs are loaded into the EMUIR register. The EMUIR is a 128-bit register. In emulation mode the register is loaded four times in a sequence (with the same JTAG instruction); and each time the data is loaded into the next word in EMUIR—Bits31–0 are loaded first, next Bits63–32 are loaded, and so on.



Writing to EMUIR when the TigerSHARC processor is not in emulator mode is an illegal operation.

When the four EMUIR loads have completed, the instructions that were loaded to the JTAG are fed into the sequencer and executed.

When a quad-word is loaded into EMUIR, it should include one or more full instruction lines. The instruction line must not continue to the next quad-word. In this case the last slots have to be filled with “no operation” (NOP) instructions. Except for the restriction of not crossing the quad-word boundary, this is similar to the assembler coding described in “Instruction Set” in the *ADSP-TS101 TigerSHARC Processor Programming Reference*.

- **EMUDAT**

This is a 48-bit register whose 32 MSBs are used to scan the Ureg of the same name. The emulator uses this register to read and set all other registers in the TigerSHARC processor.

The emulator also uses this register to access memory. There must be an instruction that uses the IALUs to generate the address for the memory access whose target or source is EMUDAT. When the emulator executes a series of memory accesses, the TigerSHARC processor should insert hold cycles to allow all of the accesses to complete (just as it does normally).

JTAG Functionality

10 SYSTEM DESIGN

Overview

This chapter takes a systems approach to describing how to use the TigerSHARC processor in your application. For that reason, we describe the use of the TigerSHARC processor in a single use implementation, in multiple TigerSHARC processor designs, as well as in testing scenarios. Hardware, software, and system design information are included.

When engineers are designing a system that includes the TigerSHARC processor, they need a number of documentation resources including:

- *ADSP-TS101 TigerSHARC Processor Hardware Reference*
- *ADSP-TS101 TigerSHARC Processor Programming Reference*
- *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*

Included in both the Analog Devices manuals and data sheet are references of source materials, such as Engineer-to-Engineer Notes or Application Notes, available on the Analog Devices Web site <http://www.analog.com>.

Overview

The TigerSHARC processor supports many system design options. The options implemented in a system may be influenced by cost, performance, and system requirements. This chapter discusses the following topics:

- “TigerSHARC Processor Pins” on page 10-3
- “Power, Reset, and Clock Input Considerations” on page 10-11
- “Reset and Boot” on page 10-17
- “Booting a Multiprocessor System” on page 10-26
- “JTAG Issues” on page 10-38
- “Resources and References” on page 10-38

Refer to Table 10-1 to locate additional information for system design.

Table 10-1. Additional Reference Material

Topic	Can be found in this document...
Power Supply Sequence	See “Power Supplies” in <i>ADSP-TS101 TigerSHARC Embedded Processor Data Sheet</i> .
Reset Sequence	See “Reset and Booting” in <i>ADSP-TS101 TigerSHARC Embedded Processor Data Sheet</i> .
Clock Domain	See “Clock Domain” in <i>ADSP-TS101 TigerSHARC Embedded Processor Data Sheet</i> .
High Frequency Design	see the Technical article <i>ADSP-TS101S MP System Simulation and Analysis</i> which can be found on the Analog Devices Web site http://www.analog.com .

TigerSHARC Processor Pins

This section describes the pins of the TigerSHARC processor and shows how these signals can be used in your system. Figure 10-1 illustrates how the pins are used in a single processor system.

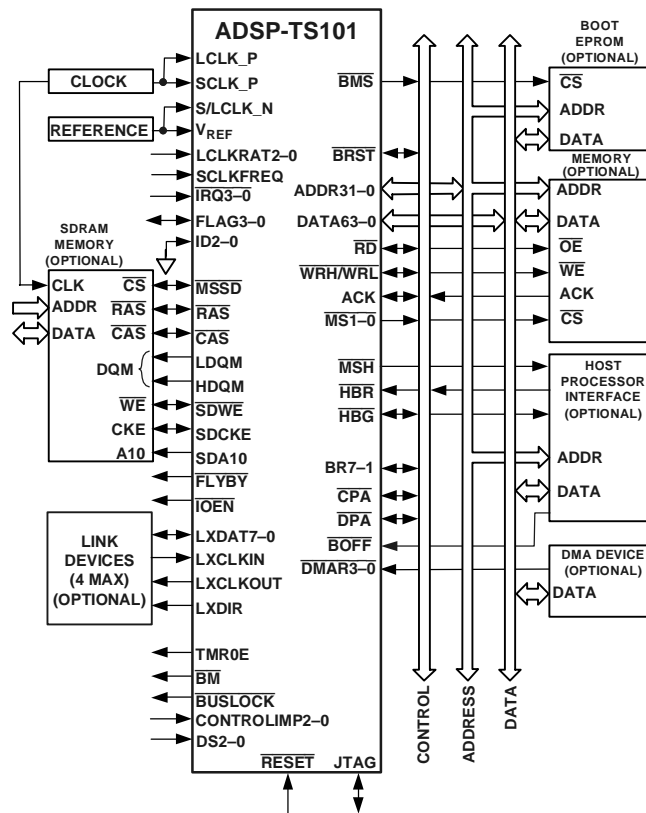


Figure 10-1. Basic TigerSHARC Processor System

Pin Definitions

Table 10-2 provides a complete list of pins in the TigerSHARC processor, listed alphabetically by pin name. Pin name, pin type, and a brief description are provided. Pin type legend:

- A = Asynchronous
- G = Ground
- I = Input
- O = Output
- o/d = Open drain output
- P = Power supply



Table 10-2 includes only a brief description of pins. For a complete description, refer to the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet* which contains the most current and detailed information about this TigerSHARC processor product.

Table 10-2. General Pin Descriptions


Signal	Type	Description
ACK	I/O/T	Acknowledge
ADDR31-0	I/O/T	Address Bus
\overline{BM}	O	Bus Master
\overline{BMS}	O/T	Boot Memory Select
\overline{BOFF}	I/O/T	Back Off
$\overline{BR7-0}$	I/O	Multiprocessing Bus Request Pins
\overline{BRST}	I/O/T	Burst
$\overline{BUSLOCK}$	O/T	Bus Lock Indication
\overline{CAS}	I/O/T	Column Address Select
CONTROLIMP2-1	I	Impedance Control
CONTROLIMP0	I	Impedance Control
\overline{CPA}	I/O (o/d)	Core Priority Access
DATA63-0	I/O/T	External Data Bus
$\overline{DMAR3-0}$	I/A	DMA Request Pins
DPA	I/A	DMA Request Pins
DS2-0	I	Digital Drive Strength Selection
\overline{EMU}	O (o/d)	Emulation
FLAG3-0	I/O/A	FLAG pins
\overline{FLYBY}	O/T	Flyby Mode
\overline{HBG}	I/O/T	Host Bus Grant
\overline{HBR}	I	Host Bus Request
HDQM	O/T	High Word SDRAM Data Mask
ID2-0	I	Multiprocessor ID


Table 10-2. General Pin Descriptions (Cont'd)

Signal	Type	Description
$\overline{\text{TOEN}}$	O/T	I/O Device Output Enable
$\overline{\text{TRQ3-0}}$	I/A	Interrupt Request
LxCLKIN	I/A	Linkx Clock/Acknowledge Input
LxCLKOUT	O	Linkx Clock/Acknowledge Output
LxDAT7-0	I/O	Linkx Data 7-0
LxDIR	O	Link0 Direction. (0 = input, 1 = output)
LCLK_N		Local Clock Reference
LCLK_P	I	Local Clock Input, DSP clock input
LCLKRAT2-0	I	LCLK Ratio
LDQM	O/T	Low Word SDRAM Data Mask
$\overline{\text{MS1-0}}$	O/T	Memory Select
$\overline{\text{MSH}}$	O/T	Memory Select Host
$\overline{\text{MSSD}}$	I/O/T	Memory Select SDRAM
NC		No Connect
$\overline{\text{RAS}}$	I/O/T	Row Address Select
$\overline{\text{RD}}$	I/O/T	Memory Read
$\overline{\text{RESET}}$	I/A	Reset
SCLK_N	I	System Clock Reference
SCLK_P	I	System Clock Input
SCLKFREQ	I	SCLK Frequency
SDA10	O/T	SDRAM Address bit 10 pin
SDCKE	I/O/T	SDRAM Clock Enable
$\overline{\text{SDWE}}$	I/O/T	SDRAM Write Enable
TCK	I	Test Clock (JTAG)

Table 10-2. General Pin Descriptions (Cont'd)

Signal	Type	Description
TDI	I	Test Data Input (JTAG)
TDO	O/T	Test Data Output (JTAG)
TMR0E	O	Timer 0 expires
TMS	I	Test Mode Select (JTAG).
$\overline{\text{TRST}}$	I/A	Test Reset (JTAG)
V _{DD}	P	VDD pins for internal logic
V _{DD_A}	P	VDD pins for analog circuits. Pay critical attention to bypassing this supply.
V _{DD_IO}	P	VDD pins for I/O buffers
V _{REF}	I	Reference voltage defines the trip point for all input buffers, except RESET, $\overline{\text{TRQ3-0}}$, $\overline{\text{DMAR3-0}}$, $\overline{\text{ID2-0}}$, $\overline{\text{CONTROLIMP2-0}}$, TCK, TDI, TMS, and $\overline{\text{TRST}}$.
V _{SS}	G	Ground pins
V _{SS_A}	G	Ground pins for analog circuits
$\overline{\text{WRH}}$	I/O/T	Write High
$\overline{\text{WRL}}$	I/O/T	Write Low

 All pins are sampled by a clock, either a system clock (SCLK) or core clock (CCLK).

 No keeper latches exist on TigerSHARC processor.

Strap Pin Function Descriptions

Some pins have alternate functions at reset. Strap options set TigerSHARC processor operating modes. During reset, the TigerSHARC processor samples the strap option pins. Strap pins have an approximately

Strap Pin Function Descriptions

100k Ω pull-down for the default value. If a strap pin is not connected to an external pull-up or logic load, the TigerSHARC processor samples the default value during reset. If strap pins are connected to logic inputs, a stronger external pull-down may be required to ensure default value depending on leakage and/or low level input current of the logic load. To set a mode other than the default mode, connect the strap pin to a sufficiently stronger external pull-up. Table 10-3 lists and describes each of the TigerSHARC processor's strap pins.

Table 10-3. I/O Strap Pins

Signal	Pin	Description
EBOOT	$\overline{\text{BMS}}$	EPROM Boot 0 = Boot from EPROM immediately after reset (default) 1 = Idle after reset and wait for an external device to boot TigerSHARC processor through the external port or a link port
IRQEN	BM	Interrupt Enable. 0 = Disable and set $\overline{\text{IRQ}}$ interrupts to level sensitive after reset (default) 1 = Enable and set $\overline{\text{IRQ}}$ interrupts to edge sensitive immediately after reset
TM1	L2DIR	Test Mode 1 0 = Required setting during reset 1 = Reserved
TM2	$\overline{\text{TMR0E}}$	Test Mode 2 0 = Required setting during reset 1 = Reserved

Pin Usage

This section suggests recommended handling of unused pins. If an application does not make use of a function, the associated signals (pins) may still require a connection.

Table 10-4. Handling of Unused Pins

Unused Inputs		Unused Outputs		Pins That Must Be Connected
Pins	Handling Method	Pins	Handling Method	Pins
ADDR31-0	nc	MS1-0	nc	LCLK_N
DATA63-0	nc	MSH	nc	LCLK_P
\overline{RD}	nc (internal ¹ pull-up)	$\overline{BUSLOCK}$	nc	LCLKRAT2-0
\overline{WRL}	nc (internal ¹ pull-up)	\overline{FLYBY}	nc	SCLK_N
\overline{WRH}	nc (internal ¹ pull-up)	\overline{IOEN}	nc	SCLK_P
ACK	pu	\overline{LDQM}	nc	SCLKFREQ
\overline{BRST}	nc (internal ¹ pull-up)	\overline{HDQM}	nc	\overline{RESET}
BR7-0	pu	\overline{SDAIO}	nc	CONTROLIMP2-0
\overline{BOFF}	pu	$\overline{LXCLKOUT}$	nc	DS2-0
\overline{HBR}	pu	LXDIR	nc	\overline{BMS} strap ² - EBOOT
\overline{HBG}	pu for ID7-1, pu or nc for ID0	\overline{EMU} ³	nc	\overline{BM} strap ² - IRQEN
\overline{CPA}	pu for ID7-1, pu or nc for ID0			TMROE strap ² - TM1
\overline{DPA}	pu for ID7-1, pu or nc for ID0			L2DIR strap ² - TM2
$\overline{DMAR3-0}$	pu			V _{DD}
\overline{MSSD}	nc (internal ¹ pull-up)			V _{DD_A}
\overline{RAS}	nc (internal ¹ pull-up)			V _{DD_IO}
\overline{CAS}	nc (internal ¹ pull-up)			V _{REF}
SDCKE	nc (internal ¹ pull-up)			V _{SS}
nc = no connection pu = pull-up to V_{DD_IO} through 10k Ω resistor is required pd = pull-down to V_{SS} through 10k Ω resistor is required				

Strap Pin Function Descriptions

Table 10-4. Handling of Unused Pins (Cont'd)

Unused Inputs		Unused Outputs		Pins That Must Be Connected
Pins	Handling Method	Pins	Handling Method	Pins
\overline{SDWE}	nc (internal ¹ pull-up)			VSS_A
FLAG3-0	nc (internal ¹ pull-down)			\overline{TRST} ³
$\overline{TRQ3-0}$	nc (internal ¹ pull-up)			
LXDATA7-0	nc			
LXCLKIN	nc			
TCK ³	pu			
TDI ³	nc (internal pull-up)			
TMS ³	nc (internal pull-up)			
nc = no connection pu = pull-up to V_{DD_IO} through 10k Ω resistor is required pd = pull-down to V_{SS} through 10k Ω resistor is required				

- 1 There is an internal 100k Ω \pm 50% pull-up or pull-down resistor on this pin.
- 2 For strap pins refer to Table 10-3 on page 10-8.
- 3 Must be pulsed or held low after power up. For more information see the EE-68 Application Note “*Analog Devices JTAG Emulation Technical Reference (2.4)*”

Pin States At Reset

The TigerSHARC processor three-states all outputs during reset, allowing these pins to get to their internal pull-up or pull-down state. Some output pins (control signals) have a pull-up or pull-down that maintains a known value during transitions between different drivers.

For complete and current information about pin states, refer to the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*.

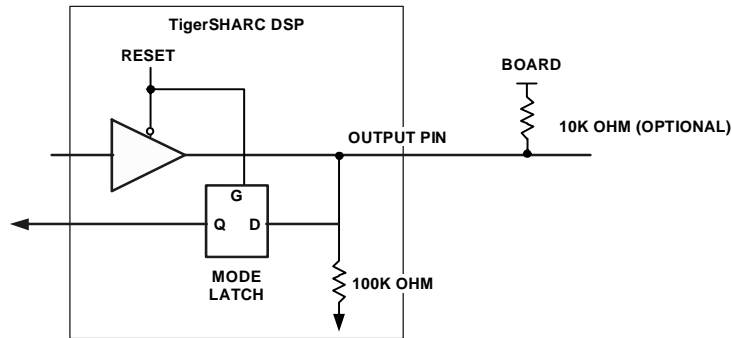


Figure 10-2. Output Pins at Reset

Power, Reset, and Clock Input Considerations

Power Supply Sequencing

For information on power supply sequencing, see “Power Supplies” in the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*.

Reset

For information on reset, see “Reset and Booting” in the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*.

Timers

This section describes how timers can be used in the TigerSHARC processor for testing and emulation purposes.

Power, Reset, and Clock Input Considerations

Timer Interrupt and FLAG I/O Examples

After reset, the FLAG pins of the TigerSHARC processor by default, become input signals (sampled by the TigerSHARC processor). As input signals, they can monitor external activities. The following instructions show how the state of the flag pins can be used to branch to different routines. Symbolic macros are defined in `defts101.h` file that is included in the VisualDSP++ Software Development Tools.

Listing 10-1. Monitor a Flag Pin

```
/*Example to show monitoring of Flag 0 pin for */
/*conditional jump when flag pin set to input*/
    if FLAG0_IN, jump function1;; /* If flag0 is set jump*/
                                /*to function1*/
    if NFLAG0_IN, jump function2;;/* If flag0 is not set*/
                                /*jump to function2*/
```

The following code sample shows how to alter the FLAG2 pin and make it an output pin (driven by the TigerSHARC processor).

Listing 10-2. Change a Flag Pin From an Input to Output Pin

```
/*Example to show setting of Flag 2 pin as an output*/
    r0 = SQCTL;; /*Read the Sequence Control Register*/
    r0 = bset r0 by SQCTL_FLAG2_EN_P;; /*Set Flag 2 as output
(bit 22)*/
    SQCTL = r0;; /* Write back to Sequence Control Register*/
```

Once a flag pin has been set as an output pin, the signal can be driven high or low by the TigerSHARC processor by setting or clearing Bits27–24 of the SQCTL register. The following code sample shows how to drive the FLAG2 signal high.

Listing 10-3. Drive a Flag Pin High

```

/*Example to show driving state of Flag 2 pin high when*/ /*set
as an output*/
    xr0 = SQCTL;;    /* Read the Sequence Control Register*/
    xr0 = bset r0 by SQCTL_FLAG2_OUT_P;;    /* Drive Flag 2 pin
high (bit 26)*/
    SQCTL = xr0;;    /*Write back to Sequence Control Register*/

```

The next example combines the two previous examples with some minor alterations to the code and shows how the FLAG2 pin is toggled when set to be an output pin instead of simply setting it. This programming change enables you to continuously toggle a programmable flag pin connected to a LED on the ADSP-TS101 EZ-KIT Lite.

Listing 10-4. Toggle a Pin

```

/*Example to show toggling of Flag 2 pin when set as an output*/
    xr0 = SQCTL;;    /*Read the Sequence Control Register*/
    xr0 = bset r0 by SQCTL_FLAG2_EN_P;;    /*Set Flag 2 as output
(bit 22)*/
    SQCTL = xr0;;    /* Write back to Sequence Control Register*/
    reset_counter:   LCO = 0x03B9AC9D;;    /*Write value to loop
counter 0*/
    wait:    if NLCOE, jump wait;;    /*If loop counter 0 not
expired, wait here*/
    xr0 = SQCTL;;    /*Read the Sequence Control Register*/
    xr0 = btgl r0 by SQCTL_FLAG2_OUT_P;;    /*Toggle the value of
Flag 2 pin (bit 26)*/
    SQCTL = xr0;;    /* Write back to Sequence Control Register*/
    jump reset_counter;;/* Repeat entire process*/

```

The following sample code shows how to set up a high priority Timer0 interrupt. The interrupt service routine contains a flag toggle routine that toggles a programmable flag pin connected to a LED on the ADSP-TS101

Power, Reset, and Clock Input Considerations

EZ-KIT Lite. The processor remains in the IDLE state until the value written to the TMRIN_x registers decrements to zero. At that point the interrupt is generated and the interrupt service routine is executed.

Listing 10-5. Set an Interrupt

```
/*Timer Example to demonstrate toggling flag within a High Priority
Timer Interrupt*/
.section program;   xr0 = IMASKH;;   /*Read contents of IMASKH
register*/
    xr0 = bset r0 by INT_TIMER0H_P;;   /*Unmask high priority
timer0 interrupt*/
    IMASKH = xr0;;   /*Write contents back to IMASKH register*/

    j0 = timer_isr_routine;;   /*Load address of ISR for Timer0
high priority interrupt*/

    IVTIMER0HP = j0;;   /* Load address of Timer0 HP ISR into IV
Table*/
    TMRINOH = 0x00000000;;   /*Load Upper 32-bits of Timer0
counter with value*/
    TMRINOL = 0x0EE6B280;;   /*Load Lower 32-bits of Timer0
counter with value*/

    xr0 = SQCTL_TMR0EN;;   /*Set Time0 run bit in the SQCTL
register*/
SQCTLST = xr0;;   /*Enable Timer0*/
    wait:   idle;;   /*Sit in IDLE until timer0 interrupt is
generated*/
    jump wait;;
timer_isr_routine:   /* Timer0 interrupt service routine*/
    xr0 = SQCTL;;   /*Read the Sequence Control Register*/
    xr0 = btgl r0 by SQCTL_FLAG2_OUT_P;;   /* Toggle the value of
Flag 2 pin (bit 26)*/
```



```
SQCTL = xr0;; /*Write back to Sequence Control Register*/  
rti(ABS);; /*Return from interrupt serviceroutine*/
```

Clock Description and Jitter

The clock signal must be free of ringing and jitter. Clock jitter can easily be introduced into a system where more than one clock frequency exists.

Clock Distribution

There must be low clock skew between DSPs in a multiprocessor cluster and between DSPs and host or memory when communicating synchronously on the external bus. The clock must be routed in a controlled-impedance transmission line that can be properly terminated, if necessary.

Figure 10-3 on page 10-16 illustrates the recommended clock distribution using point-to-point connections with optional source termination (R_T , if required for integrity). Source termination allows delays in each path to be identical. Each device must be at the end of the transmission line because it is only at that point that the signal has a single transition.

Other termination techniques are possible. It is recommended that designers simulate for signal integrity. The traces must be matched length. Clock signal traces should be in the PCB layer closest to the ground plane to keep delays stable and crosstalk low. More than one device may be at the end of the line, but the wire length between them must be short. The

Clock Description and Jitter

buffers must be in the same IC and must be specified for low jitter and low skew with respect to each other. The jitter and skew should be as small as possible because it subtracts from the margin on most specifications.

- ⊘ Never share a clock buffer IC with a signal of a different clock frequency. This introduces excessive jitter.

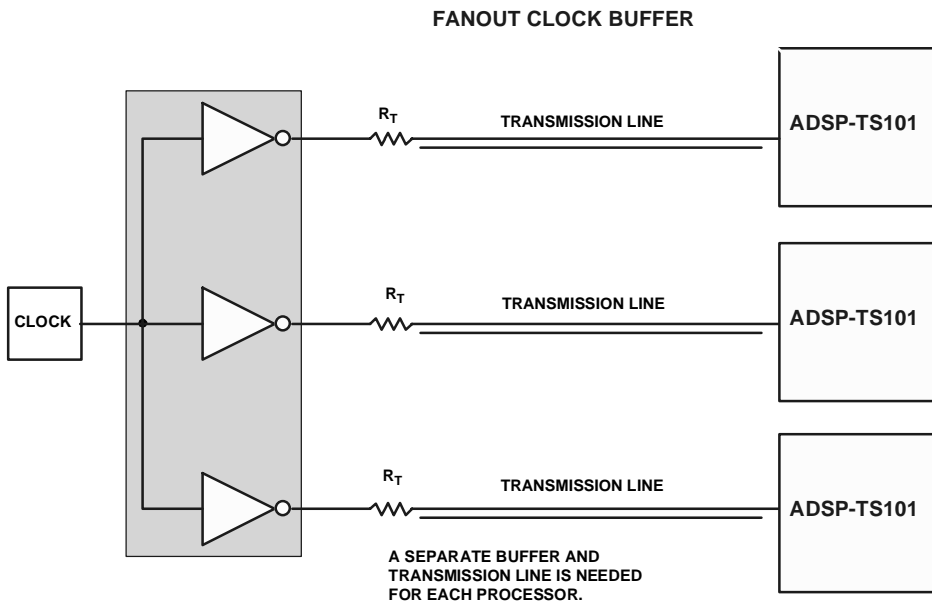


Figure 10-3. System Clock Distribution

General High Speed Clock Distribution Issues

For additional information on handling jitter, refer to *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*.

Reset and Boot

The TigerSHARC processor has three levels of reset:

- Power-up reset – After power up of the system, and after strap options are stable, the $\overline{\text{RESET}}$ pin must be asserted (low) and deasserted (high) according to the specifications detailed in the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*. During power up, $\overline{\text{TRST}}$ must also be asserted (low) to ensure proper reset of the JTAG Test Access Port (TAP). For additional details, refer to *Analog Devices Engineer- to-Engineer Note EE-68 “Analog Devices JTAG Emulation Technical Reference (2.4)”*, available on the Analog Devices Internet site: <http://www.analog.com>.
- Normal reset – For any resets following the power up reset sequence, the $\overline{\text{RESET}}$ pin must be asserted according to the specifications detailed in the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*.
- Core reset (software reset) – When setting the SWRST bit in SQCTL, the core is reset, but not the external port or I/O. (See “Sequencer Control Register – SQCTL” on page 2-16.) After a core reset, boot does not start automatically.

Hardware reset is performed by asserting the $\overline{\text{RESET}}$ pin. During reset (for hardware reset only, not core reset), all TigerSHARC processor output pins are three-stated. The strap pins are sampled during this period. The $\overline{\text{RESET}}$ pin is asynchronous. For systems that require cycle-by-cycle predictive behavior, the $\overline{\text{RESET}}$ should be asserted according to AC specifications. In this case, an integer value (2, 3, 4, 5, or 6) should be used for LCLKRAT, and the same source should be used for LCLK and SCLK pins. Refer to the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet* for the AC specifications.

Booting

Booting is the process of loading the boot loader, initializing memory, and starting the application on the target.

Software Development Tools Support for Booting

The VisualDSP++ software development environment provides support for the creation of a bootable image. This image is comprised of a loader kernel and the user's application code. VisualDSP++ includes loader kernels specific to each boot type. The boot loader kernels are 256-word assembly source code routines that perform memory initialization on the target.

The default boot loader kernels work in conjunction with the loader utility supplied with VisualDSP++ tools. The loader utility takes the user's TigerSHARC processor executable file along with the boot loader kernel executable file and produces a bootable image file. The bootable image file defines how the various blocks of TigerSHARC processor's internal memory and optional external system memory are to be initialized.

The boot loader kernel source code supplied with Analog Devices software tools may be modified. For details on the boot loader kernels refer to the *Engineer-to-Engineer Note EE-174: ADSP-TS101 Boot Loader Kernels Operation*. For complete instructions on using VisualDSP++ tools, refer to the manuals and online help provided with your tools.

Selecting the Booting Mode

The two modes for booting are Master and Slave mode. Master boot accesses an EPROM or FLASH device, and slave booting is initiated through the link port or through the external port by a host (another TigerSHARC processor, for example). The state of the external $\overline{\text{BMS}}$ pin determines the booting method. If the $\overline{\text{BMS}}$ pin is sampled low during reset, for example, it results in an EPROM or FLASH device boot.

If the $\overline{\text{BMS}}$ pin is sampled high during reset, this causes the processor to go into idle. When the processor is in the idle state waiting for a host or link boot, any signal from the host or link causes a slave mode boot.

Handling BMS

Master Mode Boot

When booting from an EPROM or FLASH device, connect an external pull-down resistor to $\overline{\text{BMS}}$.

Slave Boot Mode

When booting from either the host or link port, connect an external pull-up resistor to $\overline{\text{BMS}}$. In this case, $\overline{\text{BMS}}$ can still be used as a memory select, if necessary as the pull-up resistor can easily be overridden. If $\overline{\text{BMS}}$ is not used to access FLASH/EPROM in slave boot mode, it can alternatively be tied to V_{DD_IO} . For additional information on booting, refer to the *Engineer-to-Engineer Note EE-174: ADSP-TS101 Boot Loader Kernels Operation*.

General Boot Scenario

Regardless of which boot mode (master or slave) is used, each shares a common boot process.

1. The $\overline{\text{BMS}}$ pin determines the booting method.

Each DMA channel from which the TigerSHARC processor can boot is automatically configured for a 256-word (32-bit normal word) transfer.

Booting

2. Those first 256 instructions, called the *loader kernel*, automatically execute and perform additional DMAs to load the application executable code and data into internal and/or external memory.
3. Finally, the loader kernel overwrites itself with the application's first 256 words.

No Boot Mode

The TigerSHARC processor begins execution from an address specified in the vector table for each of the IRQs, depending on which $\overline{\text{IRQ}}$ is asserted.

Booting a Single TigerSHARC Processor

The TigerSHARC processor supports booting from:

- EPROM/flash device – This is a master mode boot
- A host, such as another TigerSHARC processor or host processor – This is a slave mode boot
- Any link port – This is a SLAVE Mode Boot


EPROM/Flash Device Boot

The EPROM boot is selected as default. The $\overline{\text{BMS}}$ pin is used as the strap option for the selection—if the $\overline{\text{BMS}}$ pin is sampled low during reset, the mode is EPROM boot.

After reset in EPROM boot, DMA channel 0 is automatically configured to perform a 256-word block transfer from an 8-bit external boot EPROM, starting at address 0 to internal memory, locations 0x00-0xFF. The DMA channel 0 interrupt vector is initialized to internal memory address 0x0. An interrupt occurs at the completion of the DMA channel 0 transfer and the TigerSHARC processor starts executing the boot loader kernel at internal memory location 0x0.

The boot loader kernel then brings in the application code and data through a series of single-word DMA transfers. Finally, the boot loader kernel overwrites itself with the application code, leaving no trace of itself in TigerSHARC processor internal memory. When this DMA process completes, the IVT entry of DMA channel 0 points to internal memory address 0, allowing the user's application code to begin execution. EPROM booting automatically uses a special 8- to 32-bit packing mode, least-significant-word first, to perform DMA reads from EPROM. Only DMA channel 0 supports this special packing mode so it must be used to boot from EPROM. Data is transferred over the data bus pins 7-0 (DATA7-0). The lowest address pins of the TigerSHARC processor should be connected to the EPROM's address lines. The EPROM's chip select should be connected to $\overline{\text{BMS}}$ and its output enable should be connected to $\overline{\text{RD}}$. Maximum EPROM address space supported is 16 Mbytes. Refer to Figure 10-4 on page 10-22.

In a multiprocessor system, the $\overline{\text{BMS}}$ output is only driven by the TigerSHARC processor bus master. This allows wire OR'ing of multiple $\overline{\text{BMS}}$ signals for a single common boot EPROM.

 The TigerSHARC processor does not implement automatic packing when writing to boot EPROM/flash memory space. Therefore the user must perform manual 32- to 8-bit unpacking when writing to flash. Access to $\overline{\text{BMS}}$ memory space is only allowed via DMA; core accesses are not possible.

For more information, see “Boot EPROM to Internal Memory” on page 10-33 and “EPROM Interface” on page 5-31.

Booting

Analog Devices supplies a default EPROM boot loader kernel (TS101_prom.asm).

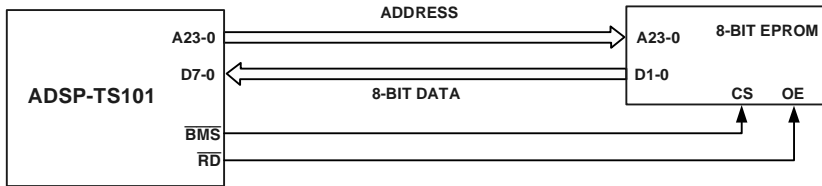


Figure 10-4. PROM Booting

Host Boot

Booting the TigerSHARC processor from a 32-bit or 64-bit host processor is performed via the data and address buses of the external port.

The $\overline{\text{BMS}}$ pin is used as the strap option for the selection. If the $\overline{\text{BMS}}$ pin is sampled high during reset, this causes the processor to go into idle and disables master mode boot DMA. When the processor is in idle state waiting for a host or link boot, any signal from the host or link causes a slave mode boot.

Host boot uses the TigerSHARC processor AutoDMA channels. Either AutoDMA channel can be used since both AutoDMA channels (AUTODMA0 and AUTODMA1) are active and initialized at reset to transfer 256 words of code and/or data into the TigerSHARC processor's internal memory block 0, locations 0x00-0xff. The corresponding DMA interrupt vectors are initialized to 0. An interrupt occurs at the completion of the DMA transfer and the TigerSHARC processor starts executing the boot loader kernel at internal memory location 0x0. It is intended that these first 256 words act as a boot loader to initialize the rest of TigerSHARC processor internal memory. The boot loader kernel then brings in the application code and data through a series of single-word DMA transfers. Finally, the boot loader kernel overwrites itself with the application

code, leaving no trace of itself in TigerSHARC processor internal memory. When this series of DMA processes completes, the IVT entry of AutoDMA channel 0 (and AutoDMA channel 1) points to internal memory address 0, allowing the user's application code to begin execution.

A host must monitor the status of the AutoDMA via the DMA Status register `DSTAT` to ensure that overrun of the AutoDMA buffer does not occur during the boot process. Buffer overrun can occur when the boot loader kernel executes time-consuming memory initializations, which delay servicing of the host writes to the AutoDMA buffer.

By default, the external bus width is configured to 32-bit width in `SYSCON` upon power up reset, therefore host as bus master must use pipelined protocol to communicate with the TigerSHARC processor. Please refer to "TigerSHARC processor Pipelined Interface" on page 6-14 for details on the pipelined protocol.

The host boot loader kernel provided by Analog Devices with the VisualDSP++ software development tools assumes the processor has been configured for normal-word transfers. Upon reset, the default state of AutoDMA is configured for quad-word accesses. In order to use the pro-

Booting

vided host boot kernel, the host must reinitialize the TigerSHARC processor's AutoDMA TCB for normal-word accesses prior to the boot process. Here are two options available to the user:

1. As described in Engineer-to-Engineer Note *EE-174: TigerSHARC processor Boot Loader Kernels Operation*, disable the active AutoDMA TCB, reconfigure the TCB to perform normal-word transfers, re-enable the TCB and proceed with host boot using normal-word transfers during the entire boot process.
2. The host device has burst transfer capability and burst transfers are desired during the entire boot process. The provided boot loader kernel must be modified and rebuilt by the user to provide burst transfer support.



If the system's host device does not have burst transfer capability, quad-word access are not be possible.



The TigerSHARC's AUTODMAs are initialized for quad-word transfers. If the host does not have control over the TigerSHARC's BRST signal, it cannot perform quad-word transfers to the AUTODMAs. Such a host requires a patch that reprograms the AUTODMA to accept normal word transfers prior to initiating the booting process.

For complete details on the patch and operation of the boot loader kernels provided by Analog Devices, refer to the Engineer-to-Engineer Note *EE-174: ADSP-TS101 Boot Loader Kernels Operation*.

Link Port Boot

This section describes how booting from a link port is performed.

The $\overline{\text{BMS}}$ pin is used as the strap option for the selection. If the $\overline{\text{BMS}}$ pin is sampled high during reset, this causes the processor to go into idle and disables master mode boot DMA. When the processor is in idle state waiting for a host or link boot, any signal from the host or link causes a slave mode boot.

Any link port can be used for booting, since all link ports are active and waiting to receive data upon power up reset or after a hard reset. Link port boot uses TigerSHARC processor's link port DMA channels. All link port DMAs are initialized to transfer 256 words to TigerSHARC processor's internal memory block 0, locations 0x00-0xFF. An interrupt occurs at the completion of the DMA transfer and the TigerSHARC processor starts executing the boot loader kernel at internal memory location 0x0. It is intended that these first 256 words act as a boot loader to initialize the rest of TigerSHARC processor internal memory. The boot loader kernel then brings in the application code and data through a series of single-word DMA transfers. Finally, the boot loader kernel overwrites itself with the application code, leaving no trace of itself in TigerSHARC processor internal memory.

When this series of DMA processes completes, the IVT entry of the link port DMA channel points to internal memory address 0, allowing the user's application code to begin execution. Analog Devices supplies a default link port boot loader kernel, (`TS101_link.asm`), with the VisualDSP++ software development tools.

The default link port loader kernel provided by Analog Devices within the VisualDSP++ software development tools uses link port #3 to perform link port booting. If another link port is required for booting the TigerSHARC processor, the boot loader kernel source code can be modified and rebuilt by the user to use any of the four link ports of the TigerSHARC processor. For additional information, refer to the Engineer-to-Engineer Note *EE-174: ADSP-TS101 Boot Loader Kernels Operation*.

Booting

Booting a Multiprocessor System

Multiprocessor systems can be booted from a host processor, from EPROM/flash, or through link ports.

The $\overline{\text{BMS}}$ pin is used as the strap option to determine the boot mode in both single and multiprocessor systems.

If all DSPs are to begin executing instructions simultaneously, all processors should include some type of software flag or hardware signal (for example, FLAG pins) to indicate when the booting of each TigerSHARC processor is complete.

Multiprocessor EPROM Booting

There are two methods of booting a multiprocessor system from an EPROM. Processors perform the following steps within these methods:

- Arbitrate for the bus
- DMA the 256-word boot stream, after becoming bus master
- Release the bus
- Execute the loaded instructions

All DSPs Boot in Turn From a Single EPROM

The $\overline{\text{BMS}}$ signals from each TigerSHARC processor may be wire-OR'ed together to drive the chip select pin of the EPROM. Each TigerSHARC processor can boot in turn, according to its priority. If all DSPs are to begin executing instructions simultaneously, then the last TigerSHARC processor must inform the other DSPs that program execution can begin when it has finished booting. Note that the other DSPs may be in an idle state at the time that the last TigerSHARC processor informs them that it has completed booting.

If the $\overline{\text{BMS}}$ pin is sampled low during reset, the boot mode is EPROM boot.

- In order to boot from a single EPROM, you must connect all $\overline{\text{BMS}}$ pins to the EPROM CS pin on the EPROM. The $\overline{\text{BMS}}$ output is only driven by the TigerSHARC processor bus master. This allows wire-OR'ing of multiple $\overline{\text{BMS}}$ signals for a single common boot EPROM.
- The EPROM loader kernel accepts multiple .DXE files and reads the ID field in SYSTAT to determine which area of EPROM to read. The TigerSHARC processor with the lowest ID number is responsible for initializing any external data that may be present in the system.

An example system that uses this “processors-take-turns” technique appears in Figure 10-5. When multiple DSPs boot from one EPROM, the DSPs can boot either identical code or different code from the EPROM. If the processors load different code, a jump table (based on processor ID) can be used to select the code for each processor.

Booting

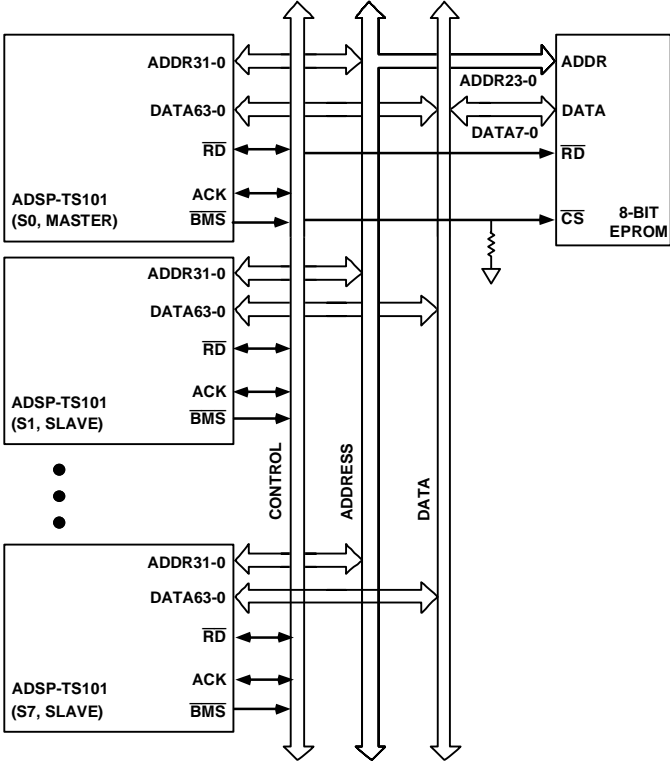


Figure 10-5. Alternating Booting from an EPROM

A Single TigerSHARC Processor Boots Other Processors

In this scenario, one TigerSHARC processor is booted, which then boots the other processors. In order to boot this single TigerSHARC processor from the EPROM, only the $\overline{\text{BMS}}$ pin of the TigerSHARC processor with $\text{ID2-0} = 000$ is connected to the chip select of the EPROM. All other processors should have their $\overline{\text{BMS}}$ signals connected to $V_{\text{DD_IO}}$ through a pull-up resistor, which configures them for host booting. Leaving all the other

processors in the idle state at startup allows the TigerSHARC processor with $ID2-0 = 000$ to become bus master and boot itself. When TigerSHARC processor with $ID2-0 = 000$ has finished booting, it can boot the remaining processors by writing to their AutoDMA channel 0 buffers via multiprocessor memory space.

An example system that uses this “one-boots-others” technique appears in Figure 10-6.

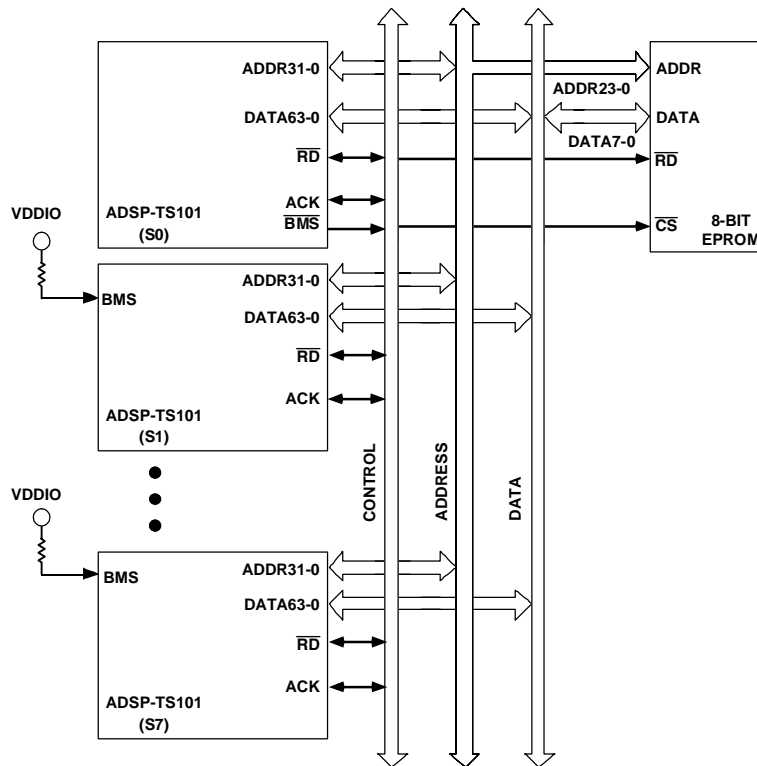


Figure 10-6. Sequential Booting from an EPROM

Booting

Multiprocessor Host Booting

If the $\overline{\text{BMS}}$ pin is sampled high during reset, this causes the processor to go into idle and disables master mode boot DMA. When the processor is in idle state waiting for a host or link boot, any signal from the host or link causes a slave mode boot.

- A host can boot each processor using the pipelined protocol and $\overline{\text{HBR}}$ and $\overline{\text{HBG}}$.
- Any TigerSHARC processor that was already booted via an EPROM, host or link Port can boot others through the external port.

Host booting multiple processors requires that the host monitor the status of the AutoDMA via the DMA status register DSTAT to ensure that overrun of the AutoDMA buffer does not occur during the boot process. Buffer overrun can occur when the boot loader kernel executes time-consuming memory initializations which delay servicing of the host writes to the AutoDMA buffer.

Recommended process for booting multiple processors via a host:

1. As described in Engineer-to-Engineer Note *EE-174: ADSP-TS101 Boot Loader Kernels Operation*, disable the active AutoDMA TCB in every processor, reconfigure the TCB to perform normal-word transfers in every processor, re-enable the TCB in every processor and proceed with host boot using normal-word transfers during the entire boot process.
2. Host performs MMS Broadcast writes to the AutoDMA channel 0 of all processors in multiprocessor system. (Recall: Writes to the broadcast region access the internal memory of all processors in the multiprocessing system. For more information on MMS space and broadcast writes, “Internal Memory Access” on page 2-7).


3. Host releases the system bus between broadcast writes if initialization of external memory by the TigerSHARC processor is necessary.
4. Host polls DMA status register (DSTAT) of each processor in the system in order to determine when the next broadcast write can occur. DSTAT polling must be performed for each processor individually since reading from broadcast MMS is not allowed. Each processor's DSTAT register indicates a status of either *Active* or *Complete* when that processor is ready to accept another normal-word broadcast write. The host must wait until all processors are ready before it can perform another broadcast write, otherwise data could be lost.
5. Steps 2 - 4 repeat until boot process completes.

Memory Initialization During Boot

During a boot process, the loader kernel initializes variables. Zeroed arrays are stored in compressed format and initialized, as required by ANSI C standards. Even if they are not explicitly initialized, large variables can potentially fill large amounts of space in object and executable files. To prevent use of large amounts of space in these files and allow the creation of non-initialized variables, use the `SHT_NOBITS` section header in the Linker Description File (LDF) to direct the linker to omit an output section from the output file. For details on LDF and `SHT_NOBITS`, see the *VisualDSP++ Linker and Utilities Manual for TigerSHARC DSPs*.

DMA Operation on Boot

If host booting from the external port and external memory is present in the system which requires initialization at boot time, the host interface should de-assert $\overline{\text{HBR}}$ between word transfers to allow the TigerSHARC processor to gain access to the bus in order to initialize external memory.

 The `SYSCON` and `SDRCN` registers must be modified if external memory is to be initialized. These registers can be modified only once after reset, and the user may need to modify these register settings in the boot loader kernel depending on system requirements.

Multiprocessor Link Port Booting

In systems where multiple processors are not connected by the parallel external bus, booting can be accomplished from a single source through the link ports. If only a daisy chain connection exists between the processors' link ports, then each TigerSHARC processor can boot the next one in turn. Any link port can be used for booting.

In link boot mode, an TigerSHARC processor that is already booted writes to any link port of another TigerSHARC processor. This is sequential booting.

DMA Operation on Boot

Booting refers to the way the processor downloads programs to the internal memory of a TigerSHARC processor after power up. For example, when the TigerSHARC processor boots from an external EPROM, the DMA controller requests data transfers from external EPROM to internal memory, and a complete block is transferred. DMA, by default, moves only 256 words to internal memory (locations `0x00000000-0x000000ff`) that comprise the boot loader (supplied with TigerSHARC processor tools). The boot loader, in turn, loads the remaining user code.

When the TigerSHARC processor boots from a link port, data is transferred to internal memory per link request. In both of these cases, the internal memory initial address is 0x00000. DMA channel 0 is used for EPROM bootstrap and any link receive DMA channel can be used for link bootstrap. The TY fields are reset in all the other channels $TCB_x DP$ registers.

Boot EPROM to Internal Memory

The transmitter TCB is programmed to start reading data from EPROM, and the receiver TCB is programmed to start writing to the internal memory. The TigerSHARC processor reads data from an 8-bit EPROM and moves the data to internal memory. Both $TCBs$ receive their values after power up reset.

If EPROM boot option is selected, the TY field is set to “Boot EPROM” in the transmitter’s $TCB DP$ register, and to “Internal Memory” in the receiver’s $TCB DP$ register. Configuration information for the $TCB DP$ register is detailed in Table 10-5 on page 10-34 and Table 10-6 on page 10-35.

DMA Operation on Boot

EPROM TCB

Upon reset, the DMA initiates a transfer by requesting external data from OFIFO:

- The TigerSHARC processor reads the data from boot EPROM into its IFIFO.
- The IFIFO gets the return address from the receiver TCB register and initiates an internal write access.
- After transferring the complete block, the DMA interrupts the core and jumps it to address 0x00000000.

Table 10-5. EPROM Boot – EPROM TCB

Transmitter TCB Configuration		
Register	Field	Description
DI		0x00000000
DX		Number of words to transfer is 256. Address modifier is set to 0x0004.
DY		0x00000000
DP	TY	Boot EPROM
DP	PR	1
DP	2DDMA	0
DP	LEN	Word
DP	INT	1
DP	DRQ	0
DP	CHEN	0
DP	CHTG	0x0
DP	CHPT	0x00000

Internal Memory TCB

Upon reset, the DMA initiates a transfer by requesting external data from OFIFO:


- The TigerSHARC processor reads the data from boot EPROM into its IFIFO.
- The IFIFO gets the return address from receiver TCB register and initiates an internal write access.
- After transferring the complete block, the DMA interrupts the core and jumps it to address 0x00000000.

Table 10-6. EPROM Boot – Internal Memory TCB

Receiver TCB Configuration		
Register	Field	Description
DI		0x00000000
DX		Number of words to transfer is 256. Address modifier is set to 0x0001.
DY		0x0000
DP	TY	Internal memory
DP	PR	1
DP	2DDMA	0
DP	LEN	Word
DP	INT	1
DP	DRQ	0
DP	CHEN	0
DP	CHTG	0x0
DP	CHPT	0x00000

Boot Link to Internal Memory

The receiver TCB is programmed to start writing into internal memory. The TigerSHARC processor link receives data and moves it into internal memory. The TCB register gets its values after power up reset.

 When writing to boot memory space (for example, to Flash memory), the byte ordering is little endian (LSB first).

Configuration information for the TCB receiver is described in Table 10-7.

Table 10-7. Link Boot – Internal Memory TCB

Receiver TCB Configuration		
Register	Field	Description
DI		0x00000000
DX		Number of words to transfer is 256. Address modifier is set to 0x0004.
DY		0x0000
DP	TY	Internal memory
DP	PR	1
DP	2DDMA	0
DP	LEN	Quad-word
DP	INT	1
DP	DRQ	1
DP	CHEN	0
DP	CHTG	0x0
DP	CHPT	0x00000

Boot AutoDMA Register to Internal Memory

The receiver TCB is programmed to start writing into internal memory. The DMA register receives data and moves it into internal memory. The TCB register gets its values after power up reset.

Upon reset, the DMA waits for a direct write into the AutoDMA register before commencing:

- Internal memory initial address is 0x000000.
- After transferring the complete block, the DMA interrupts the core and jumps it to address 0x00000000.

Configuration information for the TCB receiver is described in Table 10-8.

Table 10-8. DMA Data Register Boot – Internal Memory TCB

Receiver TCB Configuration		
Register	Field	Description
DI		0x00000000
DX		Number of words to transfer is 256. Address modifier is set to 0x0004.
DY		0x0000
DP	TY	Internal memory
DP	PR	1
DP	2DDMA	0
DP	LEN	Quad
DP	INT	1
DP	DRQ	1
DP	CHEN	0
DP	CHTG	0x0
DP	CHPT	0x00000

JTAG Issues

The Analog Devices family of emulators are tools which aid developers when testing and debugging a hardware and software system. Analog Devices has supplied an IEEE 1149.1 compliant Joint Test Action Group (JTAG) Test Access Port (TAP) on each JTAG TigerSHARC processor. The emulator uses the TAP to access the internals of the TigerSHARC processor, allowing the developer to load code, set breakpoints, observe variables, observe memory, examine registers, and so on.

For details on designing the JTAG scan path on your target system, refer to *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet* and Analog Devices Engineer-to-Engineer Note *EE-68 Analog Devices JTAG Emulation Technical Reference (2.4)*, available on the Analog Devices Internet site: <http://www.analog.com>.

Resources and References

This section identifies sources for additional information about designing and testing applications using the TigerSHARC processor.

Decoupling Capacitors and Ground Plane Recommendations

For information on using decoupling capacitors, refer to the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet*.

Signal Integrity

IBIS Models

I/O Buffer Information Specification (IBIS) models provide transistor characteristics for output drivers for use in circuit modeling.

It is recommended that users perform simulation tests to ensure proper signal integrity. For these purposes, Analog Devices supplies IBIS models. TigerSHARC processor IBIS models are available on the Analog Devices Web site <http://www.analog.com>.

For more information access:

- The Analog Devices Web site at <http://www.analog.com> and search for modeling information by entering “IBIS”
- The ANSI/EIA-656-A Home page under “Models” link <http://www.eigroup.org>

Output Pin Drive Strength Control

The pins, CONTROLIMP2-0 and DS2-0, work together to control the output drive strength of two groups of pins, the Address/Data/Control pin group and the Link pin group. CONTROLIMP2-0 independently configures the two pin groups to either the maximum drive strength or to a digitally controlled drive strength that is selectable by the DS2-0 pins. Refer to the *ADSP-TS101 TigerSHARC Embedded Processor Data Sheet* for the configuration combinations that can be selected by the CONTROLIMP2-0 pins. If the digitally controlled drive strength is selected for a pin group then the DS2-0 pins determine one of eight strength levels for that group. The drive strength selected varies the slew rate of the driver. Drive strength 0 (DS2-0 = 000) is the weakest and slowest slew rate. Drive strength 7 (DS2-0 = 111) is the strongest and fastest slew rate. The stronger drive strengths are useful for high frequency switching while the lower strengths may allow use of a relaxed design methodology. The strongest drive strengths have a

Resources and References

larger di/dt and thus require more attention to signal integrity issues such as ringing, reflections and coupling. Also a larger di/dt can increase external supply rail noise which impacts power supply and power distribution design.

The drive strengths for the EMU, CPA and DPA pins are not controllable and are fixed to the maximum level.

Separate IBIS models are supplied for each drive strength level. IBIS models are available at <http://www.analog.com>.

ADSP-TS101 Processor EZ-KIT Lite

The ADSP-TS101 processor EZ-KIT Lite provides developers with a cost-effective method for initial evaluation of the ADSP-TS101 TigerSHARC processor Family. The EZ-KIT Lite includes two TigerSHARC processors on the desktop evaluation board and fundamental debugging software to facilitate architecture evaluations via a USB-based, PC-hosted tool set. With this EZ-KIT Lite, users can learn more about Analog Devices TigerSHARC processor hardware and software development and prototype applications.

Recommended Reading References

The text *High-Speed Digital Design: A Handbook of Black Magic* is recommended for further reading. This book is a technical reference that covers the problems encountered in state-of-the-art, high frequency digital circuit design and is an excellent source of information and practical ideas. Topics covered in the book include:

- High-Speed Properties of Logic Gates
- Measurement Techniques
- Transmission Lines
- Ground Planes and Layer Stacking

- Terminations
- Vias
- Power Systems
- Connectors
- Ribbon Cables
- Clock Distribution
- Clock Oscillators

High-Speed Digital Design: A Handbook of Black Magic, Johnson and Graham, Prentice Hall, Inc., ISBN 0-13-395724-1.

Resources and References

INDEX

A

About This Document -xx

access

quad data 1-17

Acknowledge (ACK) 5-3, 5-4, 5-16, 5-23, 5-24, 5-27, 5-51

ACS instruction (Add, Compare and Select) 2-14

see also enhanced communications registers

Activ (Activate command) 6-35

Activ-to-Pre delay (tRAS) 6-2

Additional Literature -xxii

Additional Reference Material 10-2

Address Bus (ADDR) 2-2, 5-1, 5-4, 6-7, 6-9, 6-10

Address Range 7-31

address space 2-2

external memory bank space 2-2, 2-4–2-5

external multiprocessor space 2-2, 2-5–2-6, 2-44

host address space 2-2

internal address space 2-2, 2-6–2-7

addresses

indirect 1-12

relative 1-15

algorithms

digital filters 1-12

FFT 1-13

Fourier transforms 1-12

turbo-code 1-9

Viterbi 1-9

Alignments 7-30

All DSPs Boot In Turn From A Single EPROM 10-26

ALU Registers 2-13

ALU registers 2-13

Analog Devices products -xxiii

arbitration 1-16

architecture

Static Superscalar 1-6, 1-14
system 1-4

Architecture and Microarchitecture Overview 5-3

Arithmetic Logic Unit (ALU) 1-9

Arithmetic Logic Unit, *See* ALU

AutoDMA 2-44, 4-8, 7-8, 7-10, 7-12, 7-14, 7-15, 7-28, 7-30, 7-33, 7-34, 7-37, 7-38, 7-51, 10-37

AutoDMA Register Channel Operation 7-54

INDEX

- AutoDMA Register Control 7-33
- AutoDMA Register 2-44
- AutoDMA Registers 2-43
- AutoDMA Transfers 7-9
- AutoDMA0 Register 2-44
- AutoDMA1 Register 2-44
- B**
- Back Off (BOFF) 2-4, 5-39, 5-50–5-51
- Backoff 5-50
- Bank Active (ACT) Command 6-33
- Basic Transaction 5-16
- BFOTMP register 1-11
- bit
 - operations 1-11
- Bit Wise Barrel Shifter (Shifter) 1-11
- bit wise barrel shifter, *See* shifter
- bit-reversed addressing 1-13
- BMAX Current Value 2-39
- BMAX maximum cycle count 2-39, 2-41, 5-41, 5-47
- BMAX Register 2-39
- BMAX Register Bit Descriptions 2-40, 2-41
- boot 3-1, 4-2, 4-6, 10-32
 - boot by another master 3-1
 - EPROM boot (BMS) 3-1, 4-2, 5-5, 10-8, 10-32–10-34
 - link boot 3-1, 4-2, 8-7
 - no boot 3-1
- Boot AutoDMA Register to Internal Memory 10-37
- Boot EPROM to Internal Memory 10-33
- Boot Link to Internal Memory 10-36
- Boot Memory Select (BMS) 5-5, 5-32
- boot memory select (BMS) 10-8
- Booting 1-24, 10-18
- booting 1-18, 1-24, 10-18
 - EPROM 10-20
 - flash device 10-20
 - general scenario 10-19
 - link port 10-24
 - software development tools support for 10-18
- Booting a Multiprocessor System 10-26
- Booting a Single ADSP-TS101S 10-20
- branch
 - prediction 1-14, 4-10
- branch target buffer 1-7, 1-13, 1-14
- Branch Target Buffer (BTB) 2-10, 3-4
- broadcast 2-6, 2-11, 2-12, 4-8, 4-9
- Broadcast Distribution 2-7, 2-8
- Broadcast Transfer 2-11
- Broadcast Write 2-7, 2-8
- Bstop (Burst stop command) 6-34, 6-35, 6-38, 6-39
- Burst (BRST) 5-5, 5-16, 5-18, 5-50
- bus 1-16
- bus arbitration 1-20, 5-1, 5-3, 5-5,

- 5-39, 5-40, 5-41, 5-50–5-51, 7-36
 Bus Arbitration Protocol 5-41
 bus arbitration protocol 5-41–5-43
 Bus Control/Status (BIU) Register Group 2-30
 Bus Fairness — BMAX 5-47
 Bus Interface I/O Pins 5-3
 bus interface I/O pins 5-3–5-5
 Bus Interface I/O Pins (Cont'd) 5-4
 Bus Interface Unit 2-30, 3-4, 5-2
 Bus Lock 5-47
 bus lock 1-20, 2-21, 2-38, 2-41, 3-7, 3-8, 4-7, 5-39, 5-41, 5-47, 5-48, 5-51
 Bus Lock Interrupt 4-7
 Bus Lock Interrupt register *see* register
 Bus Master (BM) 4-6, 5-39
 Bus Width 5-14
 bus width 5-2, 5-14, 5-15, 5-16, 5-17, 5-35, 5-36, 6-36, 6-37, 6-40, 6-41
 Bus Width = 32 6-37, 6-41
 Bus Width = 64 6-36, 6-40
 BUSLK System Control 2-38
 BUSLK System Control Register (bus lock) 2-38, 2-41, 3-7, 5-47, 5-48
 BUSLOCK pin *see* bus lock
- C**
- CAS Before RAS 6-42
see also SDRAM
 refresh
- CAS latency 5-36, 6-2, 6-35, 6-36, 6-37, 6-38, 6-40, 6-41
 CBR *see* CAS Before RAS
 CCLK (Internal Clock) 3-2, 5-47
 Chain Insertion 7-44
 Chained DMA 7-11
 Clock Description and Jitter 10-15
 Clock Distribution 10-15
 Clock Domains 1-23
 clock signal traces 10-15
 Clocking 3-1
 CLU 1-9
 Cluster Bus 5-1
 Cluster Bus Transfers 7-7
 Column Address Strobe (CAS) 5-1, 6-2, 6-7, 6-9, 6-34, 6-38
 COM port, McBSP, *See* link ports
 Communications Logic Unit Registers 2-13
 complex numbers 1-11
 compute block 1-8
 Compute Block Register Files 2-10
 compute block register files 2-10–2-14
 ALU registers 2-13
 broadcast transfer 2-11–2-12
 compute block status registers 2-13
 merged access 2-11
 multiplier registers 2-13
 shifter registers 2-13
 unmapped compute block registers 2-12

INDEX

- Compute Block RF Groups 2-12
- compute block status registers 2-13
- Compute Blocks 1-8
- Context Switching 1-15
- context switching 1-15
- Control Bus 5-1
- Control Register (LCTLx) 8-19
- Control Signals 5-16
- Conventions -xxvii
- conventions -xxvii
- Core Controls 3-1
- Core Priority Access (CPA) 5-39, 5-41, 5-43, 5-46
- Core Priority Access CPA 5-43
- Count (XCOUNT & YCOUNT) 7-28
- customer support -xxii
- cycle counter 2-29, 3-4, 9-11
- Cycle Counter (CCNT1 0) 9-11
- Cycle Counters – CCNT0 and CCNT1 2-29

- D**
- D unit, *See* IALU or ALU
- Data Accesses 1-17
- data accesses 1-17
- Data Alignment Buffer 2-9
- data alignment buffer 1-17
 - accesses 1-17
- Data Alignment Buffer (DAB) 4-10
- Data Bus (DATA) 5-1, 5-4, 5-31, 6-7
- Data Registers 9-14
- Data Sheets -xxvi
- data sheets -xxvi
- Data Throughput Rates (Cont'd), 6-27
- DCNT Register 7-26
- DCNTCL Register 7-28
- DCNTST Register 7-28
- deadlock 5-39, 5-50–5-51
- Debug Concepts 9-3
- Debug Functionality 9-1
- Debug Register Groups 2-21
- Debug Register Groups for 0x1B Register Numbers 2-22
- Debug Register Groups for 0x3D Register Numbers 2-22
- Debug Register Groups for 0x3e Register Numbers 2-23
- Debug Resources 9-2
- Decoupling Capacitors and Ground Plane Recommendations 10-38
- DESPREAD instruction 1-9
- digital filters 1-12
- direct addressing 1-12
- Direct Memory Access 7-1
- DIx Register 7-16
- DMA 1-19, 1-22, 1-23, 2-44, 3-4, 4-2, 4-8, 5-2, 5-3, 5-7, 5-9, 5-10, 5-31, 5-34–5-37, 7-1, 8-5
 - AutoDMA 2-44, 4-8, 7-8, 7-10, 7-12, 7-14, 7-15, 7-28, 7-30, 7-33, 7-34,

- 7-37, 7-38, 7-51, 10-37
- boot 10-32
- chaining 1-22, 7-13, 7-15, 7-16, 7-18, 7-20, 7-21, 7-32, 7-33, 7-35, 7-38, 7-41–7-44, 7-46, 7-48, 7-49, 7-50, 7-52, 7-53, 7-54, 7-55, 7-56, 7-57, 7-58, 7-59, 7-60, 7-64, 7-65, 7-66
- cluster bus transfers 7-7–7-8
- control and status registers 7-23–7-31
- DMA architecture 7-11–7-12
- DMA channel 1-20, 1-22, 2-7, 2-43, 2-45, 3-6, 4-5, 4-6, 5-33, 7-7, 7-15–7-23, 7-36–7-40, 7-66–7-68, 8-5
- DMA Control register DCNT 2-46, 7-26
- DMA Control register DCNTCL 2-46, 7-28
- DMA Control register DCNTST 2-46, 7-28
- DMA Control register restrictions 7-28–7-31
- DMA Control registers 7-26–7-31
- DMA controller 1-22, 7-7–7-10, 7-32–7-49, 10-32
- DMA interrupt 2-20–2-21, 4-2, 4-5–4-6, 7-48
- DMA memory access 7-34–7-36
- DMA programming 7-14
- DMA registers 2-41, 7-15–7-31
- DMA sequences 7-48–7-49
- DMA Status register (DSTAT) 2-46, 7-23–7-25
- DMA throughput 7-66–7-68
- DMA transfers (overview) 7-33–7-34
- external port 2-43, 7-32, 7-50–7-62
- handshake mode 7-61
- link input and IFIFO DMA TCB 2-45
- link output DMA TCB 2-45
- link ports DMA 7-32, 7-63–7-65, 8-5–8-7
- link transfers 7-9
- semaphores 7-59
- TCB registers 7-15–7-23
- Transfer Control Block (TCB) registers 7-15
- two-dimensional DMA 7-10, 7-17, 7-22, 7-28, 7-32, 7-36, 7-45–7-47, 7-51, 7-52, 7-53, 7-54, 7-55, 7-56, 7-57, 7-58, 7-59, 7-60, 7-64, 7-65, 7-66, 10-34, 10-35, 10-36, 10-37
- DMA Architecture Overview 7-11
- DMA Chaining 7-41
- DMA Channel Control 7-15

INDEX

DMA Channel Prioritization 7-36
DMA Channel Priority 7-36
DMA Channel Priority 7-38
DMA Channels 7-34
DMA Control and Status Register
2-46
DMA Control and Status Register
2-46
DMA Control and Status Registers
7-23
DMA Control Register Restrictions
7-28
DMA Control Registers 7-26
DMA Controller 1-22
DMA Controller Features 7-7
DMA Controller Operations 7-32
DMA Data Register Boot
 Internal Memory TCB 10-37
DMA Interrupts 4-5, 7-48
DMA Memory Accesses 7-34
DMA Operation on Boot 10-32
DMA Priority Access (DPA) 5-39,
5-41, 5-43, 5-43–5-46
DMA Priority Access DPA 5-43
DMA Registers 2-41
DMA Request 7-30
DMA Request (DMAR) 1-22, 7-8,
7-12, 7-12–7-13, 7-32, 7-61, 7-62,
8-6
DMA Semaphores 7-59
DMA Status Register (DSTAT)
7-23
DMA Throughput 7-66
DMA Transfer Control Block Reg-

isters 7-15
DMA Transfers 7-33
DMAR I/O Pins 7-12
DPx Register 7-18
DPx Register Bit Descriptions
7-20
DSP Architecture 1-6
DSP architecture 1-6
DSP Product Information -xxiii
DSP product information -xxiii
Dual TCB Channel 7-52, 7-53
DXx Register 7-17
DYx Register 7-17

E

edge sensitive 3-6, 3-8, 4-3–4-4,
4-7, 4-16, 10-8
EMU 9-13
emulation 1-19
 EMU 9-13
 EMUIR register 4-23
 emulation debug 4-15
 emulation exception 3-3, 4-16,
 4-23
 emulation instruction 3-3
 emulation interrupt 4-3
 emulation trap 3-3, 4-10, 4-25,
 9-8
Emulation and Test Support 1-24
Emulation Debug 4-10
emulation debug 4-10
Emulation Mode 3-3
emulation mode 3-1, 3-2, 3-3–3-4,
4-10, 4-16, 9-2

- Enabling and Disabling Chaining 7-42
 - Ending a DMA Sequence 7-49
 - enhanced communications registers 2-13
 - Entering Low Power Mode 3-6
 - EP Channels (0 to 3) Type Restrictions 7-30
 - EPROM Boot
 - EPROM TCB 10-34
 - Internal Memory TCB 10-35
 - EPROM boot (BMS) 3-1, 4-2, 5-5, 10-8, 10-32–10-34
 - EPROM Interface 5-31
 - EPROM interface 5-3, 5-31–5-33
 - EPROM TCB 10-34
 - EPROM/Flash Device Boot 10-20
 - EPROM/flash device boot 10-20
 - Error Detection Mechanisms 8-17
 - exception (software interrupt) 2-21, 4-3, 4-9–4-10, 4-16, 4-23–4-25
 - Exceptions 4-23
 - external bus 1-19, 5-1–5-51, 7-7–7-8
 - External Bus and Host Interface 1-19
 - External Bus Features 5-2
 - External I/O Device to External Memory (Flyby) 7-55
 - External Memory 1-19
 - external memory 1-19, 1-21, 1-22, 2-2, 2-4–2-5, 5-3, 5-9, 7-54, 7-67
 - External Memory Bank Space 2-4
 - External Memory Bank Space 2-5
 - External Memory DMA 7-67
 - External Memory TCB 7-52, 7-56
 - External Memory TCB (Flyby) 7-58, 7-60
 - External Memory to External I/O Device (Flyby) 7-57
 - External Multiprocessor Space 2-6
 - External Port 1-19
 - external port 1-19–1-21, 1-23, 2-47, 5-7, 7-13, 7-28, 7-37, 7-38, 7-50–7-62, 7-63, 7-67, 10-8
 - External Port Configuration and Status Registers 2-41
 - External Port Configuration and Status Registers 2-41
 - External Port DMA 7-50
 - External Port DMA Control 7-32
 - External Port DMA Register 2-43
 - External Port DMA Register 2-43
 - External Port DMA Transfer Types 7-51
 - External Port Registers 2-30
 - External to Internal Memory 7-51
- F**
- FFT algorithms 1-13
 - fixed-point formats 1-8
 - FLAG (Flag pins) 3-8
 - Flag Pins 3-8
 - Flash memory interface 5-3, 5-33
 - Flyby (FLYBY) 5-3, 5-5, 5-34–5-37, 7-8, 7-55–7-57

INDEX

Flyby Mode 7-61

Flyby Transactions 5-34

Fourier transforms 1-12

G

General Boot Scenario 10-19

general boot scenario 10-19

General High Speed Clock Distribution Issues 10-16

General Pin Descriptions 10-5

Global Registers—XSTAT/YS-TAT Compute Block Status Registers 2-13

H

Handling BMS 10-19

Handling of Unused Pins 10-9

Handshake Mode 7-61

Hardware Error Operations 4-8

hardware interrupt 3-3, 4-3, 4-8, 4-12, 4-16, 4-25

Hardware Manuals -xxvi

hardware manuals -xxvi

HBG *see* Host Bus Grant

HBR *see* Host Bus Request

High Word SDRAM Data Mask (HDQM) 6-8, 6-10

Host Address Space 2-2, 2-4

Host Boot 10-22

Host Bus Grant (HBG) 1-21, 5-38, 5-40, 5-43, 5-47, 5-51

Host Bus Request (HBR) 1-21, 5-38, 5-40, 5-41, 5-43, 5-46

Host Interface 1-21, 5-48

host interface 1-21, 5-3, 5-48–5-50

I

I/O Device TCB (Flyby) 7-57, 7-59

I/O Strap Pins 10-8

IALU 1-12

IALU registers 2-14–2-15, 4-19

immediate extensions 1-14

registers 1-12

IALU Registers 2-14

IALU RF Groups 2-14

IBIS Models 10-39

IDLE (Idle state) 2-46, 3-5, 3-9, 5-2, 5-14, 5-22, 5-33

IEEE 754/854 format 1-24

IEEE standards, exceptions 1-24

IFIFO 2-45, 3-6, 5-7, 5-8, 5-9, 5-10, 7-13, 7-63, 7-67, 10-34, 10-35

ILAT Register 4-11

ILAT Registers 2-19

ILAT *see* register

IMASK Register 2-19, 4-12

IMASK *see* register

immediate extension 1-12

operations 1-14

indirect addressing 1-12

Initial Value 5-15

Instruction Address Trace Buffer (TBUF) 9-9

instruction dispatch/decode, *See* sequencer

- instruction line 1-24
- instructions
 - bit manipulation 1-11
 - immediate extension 1-14
 - quad 1-14, 1-17
- Integer Arithmetic Logic Unit (IALU) 1-12
- Integer Arithmetic Logic Unit, *See* IALU
- Intended Audience -xix
- intended audience -xix
- Internal Address Space 2-6, 2-7
- Internal and External Address Generation 7-50
- Internal Buses 1-16
- internal clock *see* CCLK
- Internal DSP Address and SDRAM Physical Connection 6-11
- internal memory 1-20, 1-21, 1-22, 2-2, 2-5, 2-6, 2-44, 5-9, 5-31, 7-1, 7-15, 7-33, 7-54, 7-67, 10-35
- Internal Memory Access 2-7
- Internal Memory and Other Internal Peripherals 1-16
- internal memory bus 7-36
- Internal Memory Bus Priority 7-36
- Internal Memory Buses 7-33
- Internal Memory DMA 7-67
- Internal Memory TCB 10-35
- Internal Memory TCB 7-53, 7-55
- Internal Memory TCB 7-54
- Internal to External Memory 7-54
- Internal Transfer 1-17
- internal transfer 1-17
- Internal/External Memory to Link 7-63
- Internal/External Memory to Link TCB 7-65
- interrupt 1-14, 1-15, 2-19, 3-6, 4-1–4-25
 - bus lock 2-21
 - DMA interrupt 2-20–2-21, 4-2, 4-5–4-6, 7-48
 - emulation interrupt 4-3
 - exception (software interrupt) 2-21, 4-3, 4-9–4-10, 4-16, 4-23–4-25
 - external 2-19, 2-21, 3-6, 10-8
 - hardware interrupt 3-3, 4-3, 4-8, 4-12, 4-16, 4-25
 - interrupt flag 4-1
 - interrupt handling 4-19–4-20
 - interrupt mask 4-1
 - interrupt types 4-3–4-4
 - IVBUSLK (Bus Lock Interrupt register) 4-7
 - link 2-20, 4-5, 4-12, 8-7
 - RETI register 4-13, 4-15, 4-16, 4-18, 9-2
 - software interrupt (exception) 2-21, 4-3, 4-9–4-10, 4-16, 4-23–4-25
 - IVSW (Software Interrupt Vector register) 4-9, 4-23
 - vector 1-21, 2-20, 2-21, 4-6, 4-7

INDEX

VIRPT (Vector Interrupt register) 4-7
interrupt flag 4-16
Interrupt Handling 4-19
interrupt handling 4-19–4-20
Interrupt I/O Pins 4-2
Interrupt Pins (IRQ) 4-6
interrupt pins *see* IRQ (Interrupt Request)
interrupt register 4-1
Interrupt Service 4-14
Interrupt Service Routine *see* ISR (Interrupt Service Routine)
Interrupt Types 4-3
interrupt vector register 4-1, 4-2
Interrupt Vector Table 4-2
Interrupt Vector Table (IVT) 2-20, 4-1, 4-2, 4-15
Interrupt Vector Table (IVT) for Group 0x38 - Register Numbers 2-20
Interrupt Vector Table (IVT) for Group 0x39 - Register Numbers 2-21
Interrupt Vector Table Register Groups 2-20
Interrupting Chaining 7-42
Interrupts 4-1, 8-7
interrupts
 timers 4-4
Interrupts Generated by On-Chip Modules 4-4
Introduction 1-1
IRQ (Interrupt Request) 2-19,

2-21, 4-2, 4-3, 4-6–4-7, 4-12, 10-8
IRQ3–0 pins 1-14
ISR (Interrupt Service Routine) 3-4, 3-8, 4-2, 4-13, 4-14–4-23
 nested ISRs 4-19–4-20
IVBUSLK register (Bus Lock Interrupt register) *see* register
IVSW (Software Interrupt Vector register) 4-9, 4-23

J

joint test action group (JTAG) 10-38
JSTAT Register Bit Descriptions 2-15
JTAG 2-29, 3-3, 4-10, 9-12
 JTAG controller 3-3
 JTAG instruction register 9-14
 JTAG reset 9-13
 Test Access Port (TAP) 3-3
JTAG and Emulation I/O Pins 9-13
JTAG Functionality 9-12
JTAG Instruction Register 9-14
JTAG Issues 10-38
JTAG port 1-24
JTAG Port Access Instruction Decode 9-15

L

L unit, *See* ALU
LCLK (Local Clock) 3-1, 5-47
LCLKRAT (Local Clock Multiplier) 3-2

- LCTLx Register Bit Descriptions 8-22
- LCTLx *see* link port
- Len Setup 7-28
- Level or Edge Interrupts 4-3
- level sensitive 4-3–4-4, 4-7, 4-8, 4-12, 10-8
- Link Architecture 8-2
- Link Boot
 - Internal Memory TCB 10-36
 - link boot 3-1, 8-7
- Link I/O Pins 8-2, 8-3
- link interrupt 2-20, 4-5, 4-12
- Link Interrupts 4-5
- link port 1-3, 1-7, 1-23, 3-4, 7-32, 7-63–7-65, 8-1, 10-8
 - architecture 8-2–8-8
 - DMA 8-5–8-7
 - error detection 8-17–8-19
 - I/O pins 8-2
 - interrupt 8-7
 - link port control register (LCTLx) 8-19–8-23
 - link port protocol 8-8–8-15
 - links registers 2-47, 2-48
 - LSTAT Link Status register 8-6, 8-17, 8-18, 8-19
 - transmission delays 8-15–8-17
 - transmitting and receiving data 8-4–8-6
- Link Port Boot 10-24
- link port boot 10-24
- Link Port Communication Protocol 8-8
- Link Port Control and Status Register 2-47
- Link Port Control and Status Register 2-47
- Link Port DMA Control 7-32
- Link Port Receive and Transmit Buffers 2-48
- Link Port Receive DMA Register 2-45
- Link Port Receive DMA Register 2-45
- Link Port Register 8-5, 8-6
- Link Port to Internal/External Memory 7-63
- Link Port Transmit DMA Register 2-45
- Link Port Transmit DMA Register 2-45
- Link Ports 1-23, 8-1
- Link Ports DMA 7-63
- Link Ports DMA Transfer Types 7-63
- Link Registers 2-47
- Link to Internal/External Memory TCB 7-64
- Link to Link TCB 7-66
- Link Transfers 7-9
- load 1-12
- local clock *see* LCLK
- Low Power Mode 3-6
- low power mode 3-6
- Low Word SDRAM Data Mask (LDQM) 6-7, 6-9
- low-power mode 3-8

INDEX

- multiprocessing systems 3-7–3-8
- LSTATx Register Bit Descriptions 8-25
- M**
- M unit, *See* multiplier
- manual
 - audience -xix
 - contents -xx
 - conventions -xxvii
 - new in this edition -xxii
 - related documents -xxiv
- master mode 10-20
- Master Mode Boot 10-19
- memory
 - internal 1-7, 1-16
- Memory Access Features 2-1
- Memory and Register Map 2-1
- Memory Initialization During Boot 10-31
- Memory Read (RD) 5-4, 5-16, 5-18, 5-27
- Memory Select SDRAM (MSSD) 6-1, 6-7, 6-9, 6-34, 6-38
- Merged Access 2-11
- Merged Distribution 2-7, 2-8
- Miscellaneous 1-23
- mode 3-1, 3-2–3-5, 9-2
 - emulation mode 3-1, 3-2, 3-3–3-4, 4-10, 4-16, 9-2
 - supervisor mode 3-1, 3-2, 3-4, 9-2
 - user mode 3-1, 3-2, 3-5, 9-2
- Mode Register Set (MRS) Command 6-30
- multichannel buffered serial port, McBSP, *See* link ports
- Multiplier Registers 2-13
- multiplier registers 2-13
- Multiply Accumulator (Multiplier) 1-11
- multiply-accumulator, *See* multiplier
- Multiprocessing 1-20, 5-38
- multiprocessing 1-18, 1-20, 1-21, 1-23, 2-2, 2-5, 2-6, 2-43, 2-44, 3-7–3-8, 4-8, 4-9, 5-1, 5-3, 5-7, 5-38–5-51, 6-42
 - configurations 1-18
 - enhanced capabilities 1-18
 - system 1-6
- Multiprocessing Bus Request (BR) 5-38, 5-40, 5-41, 5-43, 5-44, 5-47
- Multiprocessing I/O Pins (Cont'd) 5-38
- Multiprocessing Operation 6-28
- Multiprocessing Systems 3-7
- Multiprocessor EPROM Booting 10-26
- Multiprocessor Host Booting 10-30
- Multiprocessor ID (ID) 2-2, 2-5, 5-7, 5-9
- Multiprocessor Link Port Booting 10-32
- Multiprocessor Space 2-5

N

Nested Call and Interrupt 1-15
 nested calls 1-15
 No Boot Mode 10-20
 Notation Conventions -xxvii

O

OFIFO 3-6, 5-7, 5-9, 7-13, 7-57,
 7-63, 7-67, 10-34
 Operating Modes 9-2
 operation mode 3-1, 3-2–3-5, 9-2
 emulation mode 3-1, 3-2, 3-3–
 3-4, 4-10, 4-16, 9-2
 supervisor mode 3-1, 3-2, 3-4,
 9-2
 user mode 3-1, 3-2, 3-5, 9-2
 Operation Modes 3-2
 Other Interrupt Registers 4-11
 Overview 10-1

P

performance monitor 2-23, 3-4,
 9-10–9-12
 Performance Monitor Counter
 9-12
 Performance Monitor Counter –
 PRCNT 2-23
 Performance Monitor Mask –
 PRFM 2-23, 9-11
 Performance Monitors 9-10
 Pin Definitions 10-4
 Pin State During ACT Command
 6-33

Pin State During MRS Command
 6-31
 Pin State During PRE Command
 6-33
 Pin State During Read Command
 6-35
 Pin State During REF Command
 6-42, 6-43
 Pin State During Write Command
 6-39
 Pin States At Reset 10-10
 pin states at reset 10-10
 Pin Usage 10-8
 Pins 9-12
 pipe depth 5-49
 pipeline 1-7, 1-13, 1-14
 ALU 1-12
 Pipelined Protocol 5-15
 pipelined protocol 5-1, 5-15, 5-15–
 5-26
 basic transaction 5-16–5-18
 control signals 5-16
 pipe depth 5-16, 5-19, 5-22,
 6-2
 pipelined transactions 5-2,
 5-18–5-23, 5-34
 wait cycles 5-23–5-26
 Pipelining Transactions 5-18
 PMASK Register 2-19, 4-13
 PMASK *see* register
 post-modify addressing 1-12
 post-modify operator 1-12
 Power Supply Sequencing 10-11
 Power, Reset and Clock Input

INDEX

Considerations 10-11
Powering Up After Reset 6-29
power-up 10-32
 power-up reset 10-17
Pre (Precharge command) 6-2,
6-35, 6-39
Precharge (PRE) Command 6-31
Precharge to RAS Delay (tRP) Bits
6-26
Precharging 6-32
Preface -xix
pre-modify addressing 1-12
pre-modify operator 1-12
Pre-to-Activ delay (tRP) 6-2
Printed Manuals -xxv
printed manuals -xxv
Processor Family -xxiii
processor family -xxiii
Product Information -xxiii
product information -xxiii
Product Related Documents -xxiv
product-related documents -xxiv
program fetch, *See* sequencer
Program Sequencer 1-13, 4-23,
9-10
Programming Example 6-43
Programming Model 1-25
Programming—Control and Ad-
dress Pointer Registers 9-4
Purpose of This Manual -xix
purpose of this manual -xix

Q

quad access 1-17

quad accesses 1-17
Quad Data Access 1-17
Quad Instruction Execution 1-14
quad instruction execution 1-14

R

RAS to Precharge Delay (tRAS)
Bits 6-26
Read (Read command) 6-34
Read Command 6-34
Receiver Error Detection 8-18
Receiving Link Port to Link Port
7-64
Recommendations for Improving
Our Documents -xxvi
Recommended Reading Referenc-
es 10-40
recursion *see* recursion
Ref (Refresh command) 6-43
Refresh (REF) Command 6-42
refresh rate *see* SDRAM
refresh *see* SDRAM
register
 ALU registers 2-13
 BMAX maximum cycle count
 2-39, 2-41
 BUSLK System Control Reg-
 ister (bus lock) 2-38,
 2-41, 3-7, 5-47, 5-48
 compute block status registers
 2-13
 DMA registers 2-41, 7-15–
 7-31
 EMUIR register 4-23

- enhanced communications data registers 2-13
- enhanced communications registers 2-13
- IALU registers 2-14–2-15, 4-19
- ILAT registers 2-17, 2-18, 2-19, 4-4, 4-11–4-12, 4-14, 4-15, 4-16, 4-18, 7-48
- IMASK registers 2-17, 2-19, 4-3, 4-11, 4-12, 4-14, 4-15, 4-16, 7-48
- interrupt register 4-1
- interrupt vector register 4-1, 4-2
- IVBUSLK (Bus Lock Interrupt register) 4-7
- IVSW (Software Interrupt Vector register) 4-9, 4-23
- JSTAT register 2-15
- KSTAT register 2-15
- link port control register (LCTLx) 8-19–8-23
- links registers 2-47, 2-48
- LSTAT Link Status register 8-6, 8-17, 8-18, 8-19
- multiplier registers 2-13
- PMASK registers 2-17, 2-19, 3-4, 4-11, 4-13–4-14, 4-15, 4-16, 4-18, 4-19, 4-22, 4-23, 4-25
- register groups 2-9–2-47
- RETI register 4-13, 4-15, 4-16, 4-18, 9-2
- SDRCON SDRAM Configuration Register 2-36, 2-41, 5-2, 6-1, 6-2, 6-10, 6-42
- sequencer condition flag register (SFREG) 2-16, 2-18
- sequencer control register (SQCTL) 2-16, 2-18, 3-4, 3-5, 3-9, 4-2, 4-3, 4-7
- sequencer registers 2-15–2-19, 3-9
- sequencer status register (SQSTAT) 2-16, 2-18, 3-8
- shifter registers 2-13
- SYSCON System Configuration Register 2-34, 2-41, 5-2, 5-11–5-15, 5-17, 5-22, 5-27, 5-30, 5-48, 6-8
- SYSTAT System Status Register 2-31, 2-41, 4-8
- timer registers 3-9
- Trellis history registers (enhanced communications) 2-13
- unmapped compute block registers 2-12
- Ureg (Universal Register) 1-21, 2-6, 2-7, 2-9, 2-12, 2-14, 2-15, 2-29,

INDEX

- 3-3, 5-9, 8-2
- VIRPT (Vector Interrupt register) 4-7
- watchpoint control register (WPiCTL) 2-25, 9-4–9-7
- watchpoint status register (WPiSTAT) 2-27, 9-8
- Register Access Features 2-9
- register groups 2-9–2-47
 - Branch Target Buffer registers 2-10
 - bus control/status registers 2-10, 2-30, 2-44
 - compute block register files 2-10
 - debug logic registers 2-10, 2-21
 - DMA registers 2-10, 2-41
 - external port registers 2-10, 2-47
 - IALU registers 2-10, 2-14–2-15, 4-19
 - interrupt and sequencer registers 2-20
 - interrupt registers 2-10, 4-1
 - interrupt vector register groups 2-20
 - link registers 2-10, 2-47
 - sequencer registers 2-10
- Register J31
 - JSTAT 2-15
- Register K31
 - KSTAT 2-15
- Register Space 2-9
- registers
 - files 1-9
 - saving and restoring 1-15
- related documents -xxiv
- relative addresses 1-15
- Relative Addresses for Relocation 1-15
- relative addresses for relocation 1-15
- Reset 10-11
 - reset 3-4, 3-10, 4-2, 4-4, 4-5, 4-6, 4-7, 4-8, 4-9, 4-10, 5-15, 5-40, 10-11, 10-17
 - core reset 10-17
 - JTAG reset 9-13
 - RESET (Reset) 3-4, 10-17
 - Reset and Boot 8-7, 10-17
- Resources and References 10-38
- Resuming a DMA Sequence 7-49
- RETI register (Return from Interrupt) 4-13, 4-15, 4-16, 4-18, 9-2
- Return from Interrupt *see* RTI instruction (Return from Interrupt)
- Return to Normal Operation 3-8
- Returning From Interrupt 4-22
- Rotating Priority 7-39
- Row Address Strobe (RAS) 5-1, 6-7, 6-9
- RTI instruction (Return from Interrupt) 4-13, 4-15, 4-19, 4-22–4-23, 4-24, 4-25

S

S unit, *See* shifter

Scalability and Multiprocessing
1-18

scalability and multiprocessing
1-24

SCLK (System Clock) 1-23, 3-1,
3-10, 5-47

SDRAM 2-2, 2-36, 2-41, 4-8, 5-2,
6-1, 6-2, 6-7, 6-9, 6-10, 6-34, 6-38,
6-42

Precharge-to-RAS delay 6-2

RAS-to-Precharge delay 6-2,
6-35

refresh 5-1, 6-3, 6-43

refresh rate 6-3

SDRAM A10 (SDA10) 6-8,
6-9, 6-10, 6-13, 6-14,
6-15, 6-16, 6-17, 6-18

SDRAM Clock Enable (SD-
CKE) 6-8, 6-10, 6-38,
6-43

SDRAM controller 5-1

SDRAM I/O pins 6-7–6-8

SDRAM interface 5-3, 6-1,
6-43

SDRAM physical connection
6-11–6-18

SDRAM programming 6-43

SDRAM protocol 2-4, 5-1,
5-34

SDRAM Write Enable (SD-
WE) 6-8, 6-9, 6-34

SDRAM Control Register

(SDRCON) 6-19

SDRAM Controller Commands
6-30

SDRAM Controller Setup for AD-
SP-TS101S EZ-Kit Lite 6-44

SDRAM Enable - Bit0 6-22

SDRAM I/O Pins 6-7

SDRAM I/O Pins 6-7

SDRAM Interface 6-1

SDRAM Interface Throughput
6-27

SDRAM Physical Connection 6-8

SDRAM Programming 6-19

SDRAM programming 5-50

SDRCON (SDRAM Configura-
tion) (DMA 0x180484) 2-36

SDRCON SDRAM Configuration
Register 2-36, 2-41, 5-2, 6-1, 6-2,
6-10, 6-42

SDWE *see* SDRAM

Selecting the Booting Mode 10-18

selecting the booting mode 10-18

Selecting the CAS Latency Value
(CL) – Bits2-1 6-22

Selecting the Precharge to RAS
Delay (tRP) – Bits10-9 6-25

Selecting the RAS to Precharge
Delay (tRAS) – Bits13-11 6-26

Selecting the SDRAM's Page Size
(Page Boundary) – Bits5-4 6-25

Self-Refresh (SREF) Command
6-42

semaphores 1-20, 5-47, 5-51, 7-59

sequencer 1-13

INDEX

- immediate extensions 1-14
- Sequencer Control Register – SQCTL 2-16
- Sequencer Control Register Clear Bits – SQCTLCL 2-16
- Sequencer Control Register Set Bits – SQCTLST 2-16
- Sequencer Register Groups 2-15
- sequencer registers 2-15–2-19, 3-9
- Sequencer Registers 2-17
- Sequencer Status Register – SQSTAT 2-16
- Setting the Refresh Counter Value (Refresh Rate) – Bits8-7 6-25
- Setting the SDRAM Buffering Option (Pipe depth) – Bit3 6-23
- Setting Up and Starting the Chain 7-43
- Setting Up DMA Transfers 7-14
- SFREG Register 2-16
- shifter 1-11
- Shifter Registers 2-13
- shifter registers 2-13
- Signal Integrity 10-39
- Single DSP Boots Other DSPs 10-28
- Single Processor System 3-6
- Single-processor system 1-4
- Slave Boot Mode 10-19
- slave mode 10-20
- sleep mode *see* low power mode
- Slow Device Protocol 5-14, 5-27
- slow device protocol 5-2, 5-3, 5-14, 5-27–5-30, 5-34, 5-51
- Software Development Tools Support for Booting 10-18
- software development tools support for booting 10-18
- Software Exceptions 4-9
- Special Instructions 9-2
- SQCTL 2-16, 2-18, 3-4, 3-5, 3-9, 4-2, 4-3, 4-7
- SQSTAT 2-16, 2-18, 3-8
- Sref (Self-Refresh command) 6-42
- stack 4-19
 - see* recursion
- stack pointer 4-19
- stall 1-13
- standby mode *see* low power mode
- Starting a DMA Sequence 7-48
- Starting and Stopping DMA Sequences 7-48
- Static Superscalar 1-6, 1-14
- Status Register (LSTATx) 8-23
- store 1-12
- strap option 4-6, 10-8, 10-20
- strap pin 3-2, 3-10
 - function descriptions 10-7
- Strap Pin Function Descriptions 10-7
- Supervisor Mode 3-4
- Suspending a DMA Sequence 7-49
- SYSCON Programming 5-11
- SYSCON Register (DMA 0x180480) 2-34
- SYSCON System Configuration Register 2-34, 2-41, 5-2, 5-11–5-15, 5-17, 5-22, 5-27, 5-30, 5-48,

- 6-8
 - SYSTAT System Status Register
2-31, 2-41, 4-8
 - SYSTAT/SYSTATCL Register
2-31
 - system
 - development enhancements
1-4
 - multi-processor 1-6
 - single-processor 1-4
 - system architecture 5-1, 5-3–5-10
 - system clock distribution
illustrated 10-16
 - system clock *see* SCLK
 - System Design 10-1
- T**
- TCB *see* Transfer Control Block
 - Technical or Customer Support
-xxii
 - technical or customer support -xxii
 - Technical Publications Online or
on the Web -xxiv
 - technical publications online or on
the web -xxiv
 - technical support -xxii
 - Terminating Read/Write Cycles
6-32
 - Terminology 7-13
 - test access port (TAP) 10-38
 - Test Access Port (TAP) controller
9-14
 - Test Clock (TCK) 9-13
 - Test Data Input (TDI) 9-13
 - Test Data Output (TDO) 9-13
 - Test Mode Select (TMS) 9-13
 - Test Reset (TRST) 9-13
 - Timer 0 Output Pin 3-10
 - Timer Interrupt and FLAG I/O Ex-
amples 10-12
 - Timer Operations 3-9
 - Timer Registers 3-9
 - Timers 1-23, 3-9, 4-4, 10-11
 - timers 1-23, 3-9–3-10, 4-4, 10-11
 - interrupts 4-4
 - Timer 0 2-18, 2-20, 2-21, 3-9,
3-10
 - Timer 1 2-18, 2-20, 2-21, 3-9
 - timer interrupt and FLAG I/O
examples 10-12
 - timer registers 3-9
 - TMROE Timer 0 expires 3-10
 - Trace Buffer – TRCB0 to TRCB7
and TRCBPTR 2-29
 - Transfer Control Block (TCB)
2-43, 2-45, 4-6, 4-8, 5-10, 7-8,
7-10, 7-11, 7-12, 7-13, 7-15–7-23,
7-28, 7-30, 7-32, 7-33, 7-34, 7-35,
7-37, 7-40, 7-41, 7-42, 7-43, 7-45,
7-46, 7-48, 7-49, 7-50, 7-51, 7-51–
7-57, 7-61, 7-62, 7-63–7-65,
10-33, 10-34, 10-35
 - Transfer Control Block (TCB)
Registers 7-15
 - Transfer Control Blocks and Chain
Loading 7-43
 - Transmission Delays 8-15
 - Transmitter Error Detection 8-18

INDEX

Transmitting and Receiving Data
8-4

TRAP instruction (Supervisor
Trap) 4-9

tRAS *see* Activ-to-Pre delay
(tRAS)

tRP *see* Pre-to-Activ delay (tRP)

turbo-code algorithms 1-9

Two-Dimensional DMA 7-10

Two-dimensional DMA 7-45

two-dimensional DMA 7-10, 7-17,
7-22, 7-28, 7-32, 7-36, 7-45–7-47,
7-51, 7-52, 7-53, 7-54, 7-55, 7-56,
7-57, 7-58, 7-59, 7-60, 7-64, 7-65,
7-66, 10-34, 10-35, 10-36, 10-37

Two-dimensional DMA Channel
Organization 7-45

Two-dimensional DMA Operation
7-46

Type Setup — AutoDMA (Chan-
nels 12, 13) 7-30

Type Setup — EP (Channels 0 to
3) 7-30

Type Setup — Links Receive
(Channels 8 to 11) 7-29

Type Setup — Links Transmit
(Channels 4 to 7) 7-29

U

Understanding DQM Operation
6-29

Unmapped Compute Block Regis-
ters 2-12

Ureg (Universal Register)

see register

User Mode 3-5

V

Valid Width Settings 5-14

Vector Interrupt (VIRPT) 4-7

Vector Interrupt register *see* regis-
ter

VIRPT register (Vector Interrupt
register) *see* register

VisualDSP++ and Tools Manuals
-xxv

VisualDSP++ and tools manuals
-xxv

Viterbi algorithm 1-9

W

Wait Cycles 5-23

watchpoint 2-22, 2-25, 2-27, 3-3,
4-9, 9-2, 9-3–9-10

watchpoint control register
(WPiCTL) 2-25, 9-4–
9-7

watchpoint status register
(WPiSTAT) 2-27, 9-8

Watchpoint Address Pointers –
WP0L, WP1L, WP2L, WP0H,
WP1H and WP2H 2-29

Watchpoint Control – WP0CTL,
WP1CTL and WP2CTL 2-25

Watchpoint Operation 9-7

Watchpoint Status – WP0STAT,
WP1STAT and WP2STAT 2-27

Watchpoint Status (WPiSTAT)

9-8
Watchpoints 9-3
What's New in This Manual -xxii
Word Page Size 1k, 64-Bit Bus
Width 6-18
Word Page Size 256, 32-Bit Bus
Width 6-14, 6-17
Word Page Size 256, 64-Bit Bus
Width 6-13
Word page Size 512, 32-Bit Bus
Width 6-15
Word Page Size 512, 64-Bit Bus
Width 6-16
WPiCTL Register Bit Descriptions
9-6
WPiSTAT Register Bit Descriptions
9-9
Write Command 6-38
Write High (WRH) 5-4, 5-16,
5-18, 5-27
Write Low (WRL) 5-4, 5-16, 5-18,
5-27

INDEX