Information Manual

AT&T Microelectronics

DSP3210 Digital Signal Processor
The Multimedia Solution

AT&T Microelectronics

September 1991

AT&T DSP3210

Digital Signal Processor

Information Manual

A Word About Trademarks...

The following AT&T trademarks are used in this manual:

AT&T Visible Caching Operating System ™
VCOS ™

The following trademarks, owned by other entities other than AT&T, are also used in this manual:

SmartModel is a registered trademark of Logic Automation.
MS-DOS is a registered trademark of Microsoft.
Macintosh II s a registered trademark of Apple Inc.

Foreword

This manual contains detailed information regarding the design and applications of the AT&T DSP3210 Digital Signal Processor that is essential to engineers designing systems using this device. The AT&T DSP3210 Support Software Tools, AT&T DSP3210 C-Compiler, and the AT&T DSP3210 Application Library are available to aid in developing software for the devices and integrating them into system environments.

Additional information on the digital signal processor product line is available in the form of manuals, data sheets, and application notes.

AT&T DSP3210 DIGITAL SIGNAL PROCESSOR

INFORMATION MANUAL

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

1. INTRODUCTION 1-1
1.1 Digital Signal Processing With The DSP3210 1-1
1.2 The AT&T Floating-Point Digital Signal Processor Family 1-3
1.2.1 The AT&T DSP32 Digital Signal Processor 1-4
1.2.2 The AT&T DSP32C Digital Signal Processor 1-4
1.2.3 The AT&T DSP3210 Digital Signal Processor 1-5
1.3 Support Software Tools 1-6
1.4 Other Applicable Documentation 1-7
1.5 Assistance 1-7

CHAPTER 2. APPLICATIONS

2. APPLICATIONS 2-1
2.1 PC/Workstation Multimedia Applications 2-2
2.2 3-D Graphics Applications 2-3

CHAPTER 3. DSP3210 ARCHITECTURE

3. DSP3210 ARCHITECTURE 3-1
3.1 Functional Units 3-3
3.1.1 Control Arithmetic Unit (CAU) 3-3
3.1.2 Data Arithmetic Unit (DAU) 3-3
3.1.3 On-Chip Memory 3-3
3.1.4 Bus Interface 3-4
3.1.4 Serial I/O (SIO) 3-4
3.1.5 DMA Controller (DMAC) 3-4
3.1.6 Timer 3-5
3.1.6 Bit I/O (BIO) 3-5
3.2 Processor Control Features 3-5
3.2.1 Serial I/O DMA 3-5
3.2.2 Exception Processing 3-5
3.2.3 Wait-for-Interrupt 3-6
3.3 Data Types 3-6
3.3.1 32-bit 2's Complement Data Type 3-6
3.3.2 16-bit 2's Complement Data Type 3-7
3.3.3 32-bit Floating-Point Data Type 3-7
3.4 Memory Organization 3-8
3.5 Addressing Modes 3-11
3.5.1 Short Immediate 3-12
3.5.2 24-bit Immediate 3-12
3.5.3 CAU Register (rN) Direct 3-12
3.5.4 IO Register (iorN) Direct 3-12
3.5.5 DAU Register (aN) Direct 3-12
3.5.6 Memory Direct 3-12
3.5.7 Register Indirect 3-13
3.5.8 Register Indirect with Postmodification 3-13
3.5.9 Other Addressing Modes 3-13
3.6 IO Registers 3-14

CHAPTER 4. INSTRUCTION SET

4. INSTRUCTION SET 4-1
4.1 Flags 4-3
4.2 DA Instructions 4-4
4.2.1 DA Instructions - Multiply/Accumulate 4-4
4.2.2 DA Instructions - Special Functions 4-5
4.3 CA Instructions 4-6
4.3.1 CA Instructions - Control 4-6
4.3.2 CA Instructions - Arithmetic/Logic 4-8
4.3.3 CA Instructions - Data Move 4-10
4.4 DSP3210 Programming 4-12
4.4.1 Restrictions 4-12
4.4.1.1 Restriction 1 - Z-Field Pointer 4-12
4.4.1.2 Restriction 2 - CAU and IO Register Store 4-12
4.4.1.3 Restriction 3 - CAU Register Load 4-13
4.4.2 Latency Effects 4-13
4.4.2.1 Latency 1 - DA Instruction - Memory Writes 4-14
4.4.2.2 Latency 2 - Accumulator as Multiplier Input 4-14
4.4.2.3 Latency 3 - Branching 4-14
4.4.2.4 Latency 4 - Conditional Branching on DA Conditions 4-15
4.4.3 Common Programming Techniques 4-15
4.4.3.1 Subroutine Nesting 4-15
4.4.3.2 Passing Parameters 4-16
4.4.3.3 Quick Interrupt Facility 4-17
4.5 Common Signal Processing Functions 4-17
4.5.1 Standard Signal Processing Functions 4-17
4.5.2 Nonlinear Functions 4-18
4.6 Detailed Description of Instruction Set 4-20

CHAPTER 5. SIGNAL DESCRIPTION

5. SIGNAL DESCRIPTION 5-1

CHAPTER 6. BUS INTERFACE

6. BUS INTERFACE 6-1
6.1 Bus Interface Signal Descriptions 6-1
6.2 Processor Control Word 6-4
6.3 Standard Bus Cycles 6-6
6.3.1 Read Transaction (0-or-more waits) 6-7
6.3.2 Synchronous Write Transaction (0-or-more waits) 6-7
6.3.3 Read Transaction (1 to 3-or-more waits) 6-7
6.3.4 Write Transaction (1 to 3-or-more waits) 6-8
6.3.5 Idle State 6-8
6.4 Bus Error 6-8
6.4.1 Bus Error Operation 6-11
6.5 Bus Arbitration 6-12
6.5.1 Bus Arbitration Operation 6-12
6.5.2 DSP3210 BRN Functional Timing 6-13
6.6 Special Interface Capabilities 6-15
6.6.1 Page Break Detect Indicator 6-15
6.6.2 Interlocked Bus Cycle Indicator 6-15
6.6.3 Quad-word Block Move Indicators 6-16
6.6.4 Multiplexed Address and Data Bus 6-17
6.6.5 Use of the Data Latch Enable 6-18

CHAPTER 7. PROCESSOR STATES

7. PROCESSOR STATES 7-1
7.1 Normal Instruction Cycle 7-1
7.2 Wait-States 7-1
7.3 Direct Memory Access 7-2
7.4 Wait-for-Interrupt Mode 7-3
7.5 Instruction Sequencing Exceptions 7-3
7.5.1 Interrupt Exceptions 7-6
7.5.1.1 Quick Interrupt 7-8
7.5.2 Error Exceptions 7-8
7.5.2.1 Double Error Exceptions 7-9
7.5.3 Description of Error Exception Sources 7-10
7.5.3.1 Bus Error 7-10
7.5.3.2 Illegal Opcode 7-10
7.5.3.3 Address Error 7-10
7.5.3.4 DAU Underflow/Overflow 7-10
7.5.3.5 NAN 7-11
7.5.4 Reset 7-11
7.5.4.1 Reset Timing 7-12

CHAPTER 8. DATA PROCESSING FUNCTIONAL UNITS

8. DATA PROCESSING FUNCTIONAL UNITS 8-1
8.1 Control Arithmetic Unit (CAU) 8-1
8.1.1 CAU Programming Model 8-1
8.1.2 CAU Functional Description 8-3
8.1.3 CAU Internal Operation 8-5
8.2 Data Arithmetic Unit (DAU) 8-7
8.2.1 DAU Programming Model 8-7
8.2.2 DAU Functional Description 8-8
8.2.3 Floating-Point Arithmetic Precision 8-10
8.2.4 Data Type Conversions 8-12
8.2.4.1 IEEE Standard Floating-Point 8-12
8.2.4.2 Companded 8-13
8.2.4.3 8-bit Unsigned Integer 8-13
8.2.4.4 16-bit Integer 8-13
8.2.4.5 32-bit Integer 8-14
8.2.4.6 Reciprocal Seed 8-14
8.2.5 Rounding Modes 8-14
8.2.6 DAU Internal Operation 8-15

CHAPTER 9. PERIPHERAL FUNCTIONAL UNITS

9. PERIPHERAL FUNCTIONAL UNITS 9-1
9.1 Serial I/O 9-1
9.1.1 SIO Programming Model 9-1
9.1.2 Pin Descriptions 9-5
9.1.3 Serial Input Port 9-5
9.1.3.1 Configuring the Serial Input Port 9-6
9.1.3.2 Serial Input Timing Specifications 9-7
9.1.4 Serial Output Port 9-9
9.1.4.1 Configuring the Serial Output Port 9-10
9.1.4.2 Serial Output Timing Specifications 9-10
9.1.5 SIO On-Chip Clock Generator 9-12
9.1.5.1 Configuring the Clock Generator 9-12
9.1.5.2 Clock Generator Timing Specifications 9-12
9.1.5.3 Loss of Sanity 9-13

9.1.5.4 Loss of Sync 9-13
9.1.5.5 Frame Boundary Condition 9-14
9.1.6 Handling SIO activity 9-15
9.1.6.1 Program controlled SIO 9-15
9.1.6.2 Interrupt controlled SIO 9-16
9.1.6.3 DMA controlled SIO 9-17
9.2 DMA Controller (DMAC) 9-18
9.2.1 DMAC Programming Model 9-18
9.2.2 DMAC Functional Description 9-18
9.2.3 Serial Output DMA 9-19
9.2.4 Serial Input DMA 9-20
9.3 Timer 9-21
9.3.1 Timer Programming Model 9-21
9.3.2 Timer Functional Description 9-21
9.3.3 Timer Timing Specifications 9-24
9.4 Bit I/O (BIO) 9-24
9.4.1 BIO Programming Model 9-25
9.4.2 BIO Functional Description 9-26
9.4.3 BIO Timing Specifications 9-26

CHAPTER 10. INSTRUCTION SET ENCODINGS

10. INSTRUCTION AND REGISTER ENCODINGS 10-1
10.1 Instruction Set Encodings 10-1
10.2 Register Encodings 10-6

CHAPTER 11. PHYSICAL SPECIFICATIONS

11. PHYSICAL SPECIFICATIONS 11-1
11.1 Pin Assignments 11-2

CHAPTER 12. ELECTRICAL SPECIFICATIONS

12. ELECTRICAL AND TIMING SPECIFICATIONS 12-1
12.1 Electrical Specifications 12-1
12.1.1 Maximum Ratings 12-1
12.2 Timing Specifications 12-2

APPENDIX A

GLOSSARY

INDEX

DSP3210 PROGRAMMER'S REFERENCE CARD

LIST OF FIGURES

Figure 1-1 Dynamic Range of Fixed- and Floating-Point DSPs 1-2
Figure 1-2 Precision of Fixed- and Floating-Point DSPs 1-3
Figure 1-3 AT&T Floating-Point DSP Family Evolution 1-3
Figure 2-1 Typical PC Multimedia System Configuration 2-2
Figure 2-2 Typical 3-D Graphics System Configuration 2-3
Figure 3-1 DSP3210 Block Diagram 3-2
Figure 3-2 DSP32C Internal 32-bit Floating-Point Format 3-7
Figure 3-3 Memory Addressing 3-9
Figure 3-4 Memory Map 3-10
Figure 3-5 IO Memory Map 3-11

Figure 4-1 Register Programming Model 4-2
Figure 5-1 DSP3210 Signal Groups 5-1
Figure 6-1 External Memory Select Signaling and Write Data Duplication 6-2
Figure 6-2 PCW Register Encoding 6-5
Figure 6-3 Bus Interface State Diagram 6-6
Figure 6-4 Synchronous Bus Protocol (0-or-more wait-states) 6-9
Figure 6-5 Bus Protocol (1-or-more wait-states) 6-9
Figure 6-6 Bus Protocol (2-or-more wait-states) 6-10
Figure 6-7 Bus Protocol (3-or-more wait-states) 6-10
Figure 6-8 Read Cycle Retry 6-11
Figure 6-9 Bus Arbitration Example 6-13
Figure 6-10 Quad-word Block Move Timing 6-17
Figure 6-11 Multiplexing the Address Bus and Data Bus 6-17
Figure 6-12 DLE Timing 6-18
Figure 7-1A Pipeline for a Single DA Instruction 7-1
Figure 7-1B Full Pipeline for Multiple DA Instructions 7-1
Figure 7-2 Instruction Cycle with Conflict and External Wait-States 7-2
Figure 7-3 DMA Cycle Timing 7-3
Figure 7-4 Processing Levels 7-4
Figure 7-5 Exception Vector Table 7-5
Figure 7-6 Interrupt Exception Timing 7-7
Figure 7-7 Example of Error Exception Process 7-9
Figure 7-8A Power-On Initiates the Reset Sequence 7-12
Figure 7-8B RESTN Initiates the Reset Sequence 7-13
Figure 8-1 CAU Programming Model 8-1
Figure 8-2 CAU Register Usage 8-2
Figure 8-3 Block Diagram of the CAU 8-4
Figure 8-4 CAU Internal Pipeline Timing - CA Instruction 8-6
Figure 8-5 CAU Internal Pipeline Timing - DA Instruction 8-6
Figure 8-6 DAU Programming Model 8-7
Figure 8-7 Block Diagram of the DAU 8-9
Figure 8-8 DAU Floating-Point Formats 8-10
Figure 8-9 Precision of DAU Operations (mantissas only) 8-11
Figure 8-10 IEEE Standard 32-bit Floating-Point Format 8-12
Figure 8-11 Floating-Point to Integer Mapping for All Rounding Modes 8-15
Figure 8-12 DAU Internal Pipeline Timing 8-17
Figure 9-1 SIO Block Diagram 9-3
Figure 9-2 Serial Input Port 9-7
Figure 9-3 Serial Input Protocols (IIC=0) 9-8
Figure 9-4 Serial Input Timing (IIC=0) 9-8
Figure 9-5 Serial Output Port 9-9
Figure 9-6 Serial Output Protocol 9-10
Figure 9-7 Serial Output Timing 9-11
Figure 9-8 On-Chip Clock Generator 9-11
Figure 9-9 Clock Generator Signals 9-13
Figure 9-10 Clock Generation Timing (IIC=0) 9-14
Figure 9-11 Frame Boundary Condition 9-15
Figure 9-12 DMAC Block Diagram 9-19
Figure 9-13 Timer Block Diagram 9-22
Figure 9-14 Timer Timing Specifications 9-24
Figure 9-15 Bit I/O Bit Slice Diagrams 9-26
Figure 9-16 BIO Timing Specifications 9-27
Figure 12-1 CKI Timing Specifications 12-3
Figure 12-2 Synchronous Bus Interface Timing 12-6
Figure 12-3 Asynchronous Bus Interface Timing 12-7
Figure 12-4 Bus Arbitration Timing 12-8
Figure 12-5 Serial Input Timing 12-9

Figure 12-6 Serial Output Timing 12-10
Figure 12-7 Serial Clock Generator Timing 12-12
Figure 12-8 Bit I/O Timing 12-13
Figure 12-9 Interrupt Timing 12-14
Figure 12-10 RESTN Test Load Circuit 12-15
Figure 12-11 Power-on Reset Timing 12-16
Figure 12-12 Externally Controlled Reset and ZN Timing 12-16

LIST OF TABLES

Table 1-1 DSP3210 Features 1-6
Table 3-1 DSP3210 Features/Benefits 3-1
Table 3-2 Addressing Modes Allowed in Each Instruction Type 3-12
Table 4-1 DSP3210 Flags 4-3
Table 4-2 DA Multiply/Accumulate Instructions 4-4
Table 4-3 Replacement for DA Multiply/Accumulate Instructions 4-5
Table 4-4 DA Special Function Instructions 4-5
Table 4-5 Replacement for DA Special Function Instructions 4-5
Table 4-6 CA Control Instructions 4-6
Table 4-7 Replacement for COND 4-7
Table 4-8 CA Arithmetic/Logic Instructions 4-9
Table 4-9 CA Data Move Group Instructions 4-11
Table 4-10 Replacement for CA Data Move Instructions 4-11
Table 5-1 Pin Summary 5-1
Table 5-2 Pin Descriptions 5-2
Table 5-3 Signal Summary 5-4
Table 6-1 BGN-BERRN-SRDYN Codes and Results 6-8
Table 7-1 Priority of Exception Processing 7-5
Table 7-2 emr Register Encoding 7-6
Table 7-3 Registers Affected by Error Exceptions Other Than Reset 7-9
Table 7-4 Registers Affected by Reset 7-11
Table 8-1 ps Register 8-2
Table 8-2 CAU Operations 8-3
Table 8-3 dauc Register 8-8
Table 8-4 ctr Register 8-8
Table 9-1 ioc Register 9-4
Table 9-2 Pin Descriptions 9-5
Table 9-3 DMAC Register Encoding 9-20
Table 9-4 TCON Register Encoding 9-23
Table 9-5 bioc Register Encoding 9-25
Table 9-6 BIO Register Encodings (Writing and Reading) 9-25
Table 10-1 Instruction Encodings 10-1
Table 10-2 CA Instruction Field Encodings 10-3
Table 10-3 DA Instruction Field Encodings 10-5
Table 10-4 ps Register Encoding 10-6
Table 10-5 dauc Register Encoding 10-6
Table 10-6 ctr Register Encoding 10-7
Table 10-7 emr Register Encoding 10-7
Table 10-8 pcw Register Encoding 10-7
Table 10-9 ioc Register Encoding 10-9
Table 10-10 dmac Register Encoding 10-10
Table 10-11 tcon Register Encoding 10-11
Table 10-12 bioc Register Encoding 10-12
Table 10-13 bio Register Encoding 10-12
Table 11-2 Pin Descriptions 11-2

Table 12-1 Electrical Requirements and Characteristics 12-1
Table 12-2 Timing Requirements for CKI 12-3
Table 12-3 Timing Requirements for Synchronous Bus Interface Inputs 12-3
Table 12-1 Electrical Requirements and Characteristics 12-1
Table 12-2 Timing Requirements for CKI 12-3
Table 12-3 Timing Requirements for Synchronous Bus Interface Inputs 12-3
Table 12-4 Timing Characteristics for Synchronous Bus Interface Outputs 12-4
Table 12-5 Timing Relationships for Synchronous Bus Interface Operation 12-5
Table 12-6 Timing Relationships for Asynchronous Bus Interface Operation 12-7
Table 12-7 Timing Characteristics for Bus Arbitration 12-8
Table 12-8 Timing Requirements for Serial Inputs 12-9
Table 12-9 Timing Requirements for Serial Outputs 12-10
Table 12-10 Timing Characteristics for Serial Outputs 12-10
Table 12-11 Timing Requirements for Serial Clock Generation 12-11
Table 12-12 Timing Characteristics for Serial Clock Generation 12-11
Table 12-13 Timing Requirements for Bit I/O 12-13
Table 12-14 Timing Characteristics for Bit I/O 12-13
Table 12-15 Timing Requirements for Interrupts 12-14
Table 12-16 Timing Characteristics for Interrupts 12-14
Table 12-17 Timing Requirements for Reset 12-15
Table 12-18 Timing Characteristics for Reset and ZN 12-15

LIST OF LISTINGS

Listing 9-1 SIO Interrupt Example 9-15
Listing 9-2 Input DMA Example 9-16

Chapter 1

Introduction

Chapter 1. Introduction

1. INTRODUCTION 1-1
1.1 Digital Signal Processing With The DSP3210 1-1
Figure 1-1 Dynamic Range of Fixed- and Floating-Point DSPs 1-2
Figure 1-2 Precision of Fixed- and Floating-Point DSPs 1-3
1.2 The AT&T Floating-Point Digital Signal Processor Family 1-3
Figure 1-3 AT&T Floating-Point DSP Family Evolution 1-3
1.2.1 The AT&T DSP32 Digital Signal Processor 1-4
1.2.2 The AT&T DSP32C Digital Signal Processor 1-4
1.2.3 The AT&T DSP3210 Digital Signal Processor 1-5
Table 1-1 DSP3210 Features 1-6
1.3 Support Software Tools 1-6
1.4 Other Applicable Documentation 1-7
1.5 Assistance 1-7

1. INTRODUCTION

The AT&T DSP3210 brings the power of floating-point signal processing to personal computers and workstations opening a wide range of multimedia applications. From its conception, the DSP3210 has been engineered with a single focus: enabling advanced multimedia functions on personal computers and workstations. Based on AT&T's very successful DSP32C architecture, the DSP3210 breaks new ground in its ability to be easily integrated into personal computer and workstation system designs. Particular attention was paid to primary bus interfacing; the DSP3210 is compatible with both Intel and Motorola microprocessor signalling. This allows designers to easily create low-cost systems using the DSP3210 as a bus-master device. A full bus-level *SmartModel* * of the DSP3210 is offered by Logic Automation for system simulation of designs incorporating the DSP3210.

In addition to its optimizing C compiler and assembly language support tools, the DSP3210 features a powerful real-time operating system, the *AT&T Visible Caching Operating System™* (*VCOS™*), which supports both multi-tasking and multi-processing. The VCOS operating system is an open development environment that allows the DSP3210 to use existing system memory in PCs and workstations rather than requiring expensive dedicated SRAM. Complete real-time debugging tools are included with the VCOS operating system to speed both application and algorithm development. By separating the application and algorithm development phases of multimedia software development, development schedules are simplified and shortened . The VCOS operating system includes its own multimedia function library complete with speech processing, speech recognition, graphics, music processing, and modem functions. In addition to new application software, the DSP3210 has access to the large base of existing DSP32C application software.

This document is a reference guide for the DSP3210 device. It describes the architecture, instruction set, and interfacing specifications of the device. The information necessary to write programs for the DSP3210 is also given in this manual. Separate documents are available with detailed information on the software and hardware development tools. No previous experience with digital signal processors is necessary to use this document, but an understanding of digital signal processing concepts and assembly language programming on microprocessors or minicomputers is recommended.

1.1 Digital Signal Processing With The DSP3210

A key element shared by all DSP applications is a large number of repetitive mathematical operations combined with extreme memory bandwidth requirements. To perform floating-point arithmetic operations, conventional microprocessors and co-processors use software routines and/or microcode. Newer RISC- and CISC-based microprocessors offer much higher floating-point performance but lack the memory bandwidth and signal processing architecture required for DSP applications. Microprocessors also include many functions unnecessary for signal processing resulting in an unacceptable price/performance ratio for high-volume, low-cost signal processing applications.

The DSP3210 architecture combines a hardware floating-point arithmetic unit with an architecture that supports 4 memory accesses per instruction cycle, allowing the device to perform up to 33 million floating-point operations per second (with a clock rate of 66.7 MHz). This raw floating-point performance is combined with the DSP3210's signal processing architecture to enable real-time applications such as speech compression/recognition/synthesis, 3D graphics transformations, high-quality music coding/decoding, image processing, and real-time handwriting analysis/recognition. New functions can easily be added to DSP3210-based systems with the addition of application software. Multiple DSP3210s can be combined easily to increase system performance. (AT&T's VCOS real-time operating system directly supports multiple DSP3210 implementations.)

The use of floating-point arithmetic in the DSP3210 simplifies application development because it eliminates the effects of scaling, normalization, and overflow (which complicate development when using a fixed-point device). Also, many existing algorithms developed for use on large computers and array processors (using floating-point arithmetic) can be easily converted for use on the DSP3210. Program development is typically 3 to 10 times faster on the DSP3210 than similar program development on fixed-point processors. This is especially critical in multimedia applications where time-to market is a critical parameter, and a large percentage of application development is performed by third-party application developers. In addition to ease of program development, 32-bit floating-point is often required for advanced algorithm implementations.

* *SmartModel* is a registered trademark of Logic Automation.

Early in the development of a particular application, one must determine the dynamic range and precision requirements for the given algorithm. The following is a comparison of the dynamic range and precision attainable for various data word sizes. The floating-point representation of a number in the DSP3210 consists of a 24-bit mantissa and an 8-bit exponent. The magnitude of the mantissa is normalized to lie between 1 and 2. Thus, the range for the magnitude of a floating-point number N is approximately:

$$|N| = (1 \times 2^{-127}, 2 \times 2^{127}) \text{ or } (5.9 \times 10^{-39}, 3.4 \times 10^{38}),$$

providing an overall dynamic range in excess of 1500 dB.

Figure 1-1 shows the dynamic range for DSP3210 floating-point numbers as compared with the dynamic range of 16- and 24-bit fixed-point devices.

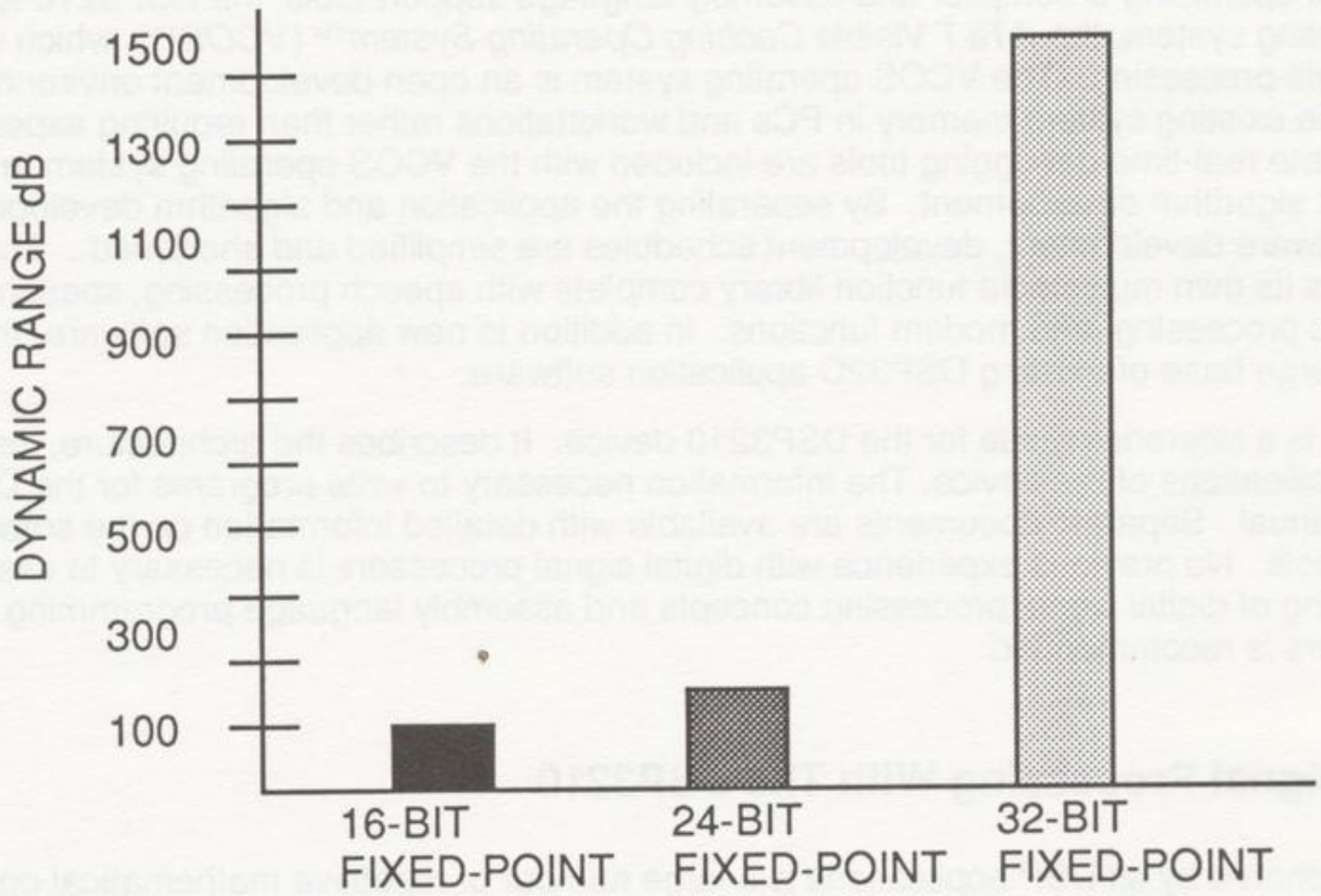


Figure 1-1 Dynamic Range of Fixed- and Floating-Point DSPs

The precision of a floating-point number is determined by the size of the mantissa. The DSP3210 floating-point format provides 24 bits of precision at all magnitudes within the range of numbers represented by the 24-bit mantissa and 8-bit exponent. (Note that the DSP3210 also has eight extra bits, the guard bits, in the mantissa of the 40-bit floating-point adder and accumulators. These guard bits provide added precision during accumulation operations). In contrast, the precision of a fixed-point format is a function of its word size and the magnitude of the given number. Figure 1-2 shows the bits of precision versus magnitude of a number for the DSP3210 floating-point format, as well as 16- and 24-bit fixed-point formats.

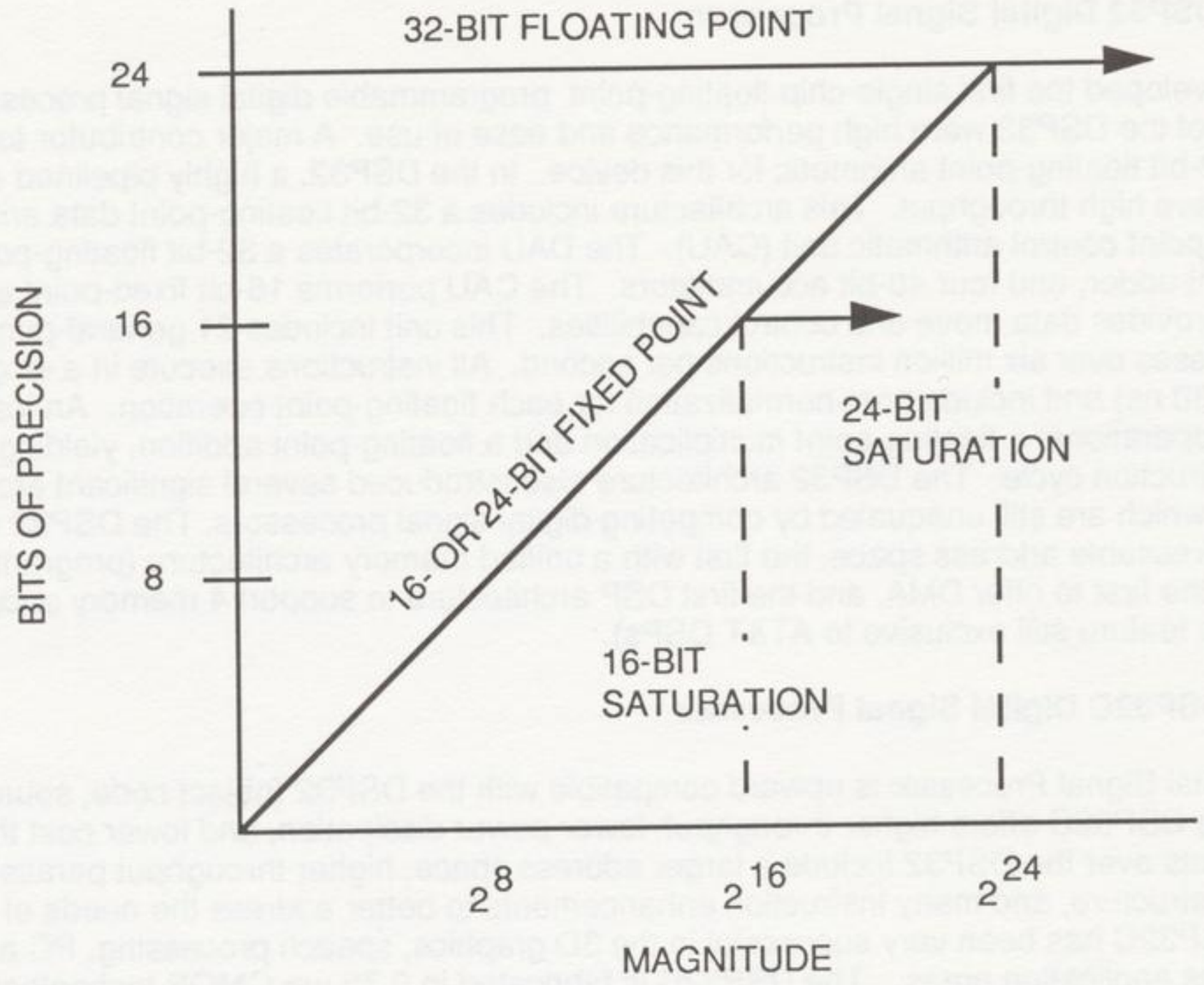


Figure 1-2 Precision of Fixed- and Floating-Point DSPs

1.2 The AT&T Floating-Point Digital Signal Processor Family

AT&T was the first to produce floating-point digital signal processors and has continued to play a leadership role in the floating-point marketplace. Figure 1-3 shows the evolution of the AT&T family of floating-point digital signal processors.

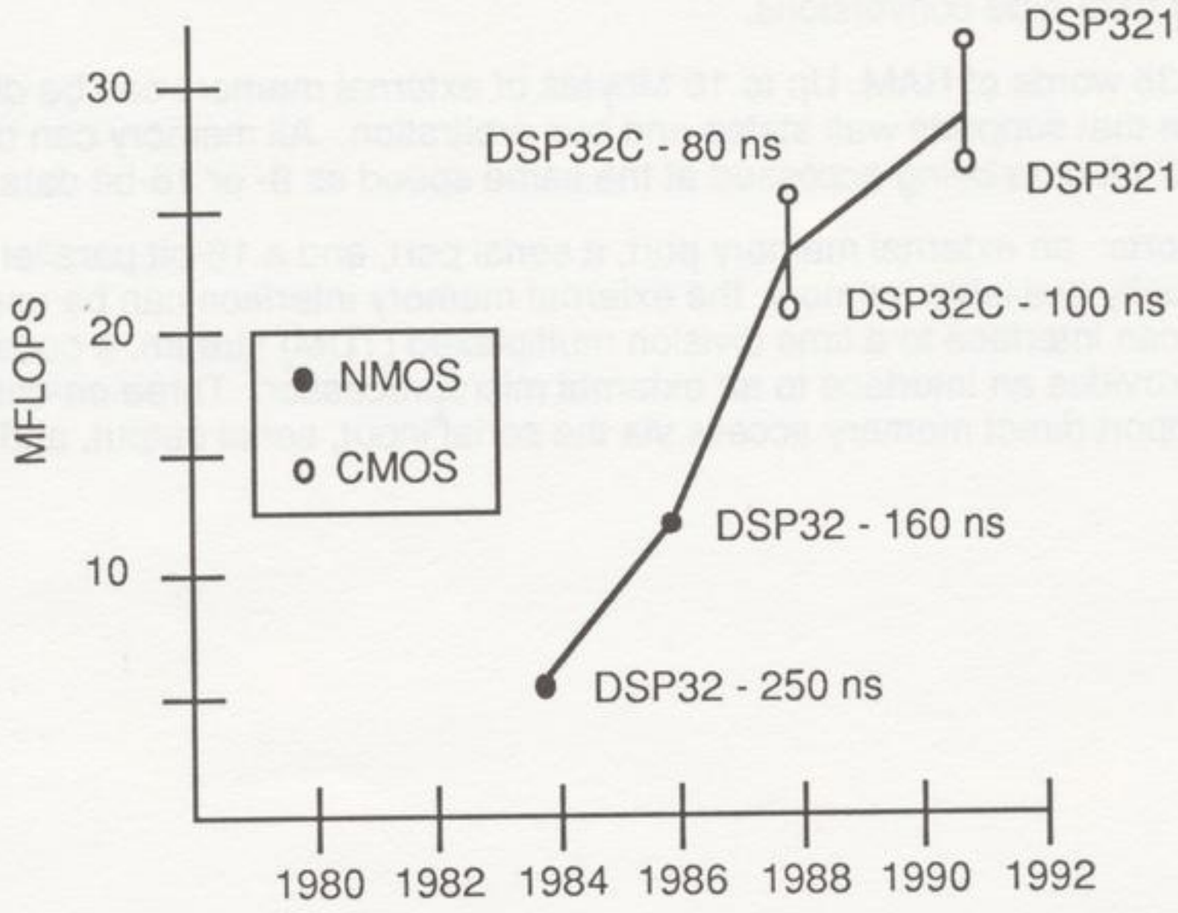


Figure 1-3 AT&T Floating-Point DSP Family Evolution

1.2.1 The AT&T DSP32 Digital Signal Processor

In 1984, AT&T developed the first single-chip floating-point programmable digital signal processor, the DSP32. The design goals of the DSP32 were high performance and ease of use. A major contributor to these goals was the selection of 32-bit floating-point arithmetic for this device. In the DSP32, a highly pipelined architecture allows the device to achieve high throughput. This architecture includes a 32-bit floating-point data arithmetic unit (DAU) and a 16-bit fixed-point control arithmetic unit (CAU). The DAU incorporates a 32-bit floating-point multiplier, a 40-bit floating-point adder, and four 40-bit accumulators. The CAU performs 16-bit fixed-point arithmetic and logic operations, and provides data move and control capabilities. This unit includes 21 general-purpose registers. The DSP32 processes over six million instructions per second. All instructions execute in a single cycle (four clock periods or 160 ns) and include post-normalization for each floating-point operation. An instruction may have two floating-point operations: a floating-point multiplication and a floating-point addition, yielding two floating-point operations per instruction cycle. The DSP32 architecture also introduced several significant architectural features, many of which are still unequaled by competing digital signal processors. The DSP32 was the first DSP to offer a byte-addressable address space, the first with a unified memory architecture (program and data share the same space), the first to offer DMA, and the first DSP architecture to support 4 memory accesses per instruction cycle (a feature still exclusive to AT&T DSPs).

1.2.2 The AT&T DSP32C Digital Signal Processor

The DSP32C Digital Signal Processor is upward compatible with the DSP32 (object code, source code, and pin compatibility). The DSP32C offers higher throughput, lower power dissipation, and lower cost than the DSP32. Major enhancements over the DSP32 include a larger address space, higher throughput parallel I/O, a fully vectored interrupt structure, and many instruction enhancements to better address the needs of DSP systems designers. The DSP32C has been very successful in the 3D graphics, speech processing, PC add-on card, and telecommunications application areas. The DSP32C is fabricated in 0.75 µm CMOS technology and is packaged in standard 164-pin and 68-pin (microcomputer version) plastic packages as well as a 133-pin ceramic PGA package.

The DSP32C's two execution units, the control arithmetic unit (CAU) and the data arithmetic unit (DAU), are used to achieve the high throughput of 12.5 million instructions per second. The CAU performs 16- or 24-bit integer arithmetic and logic operations, and provides data-move and control capabilities. This unit includes 22 general-purpose registers. The DAU performs 32-bit floating-point arithmetic for signal processing functions. It includes a 32-bit floating-point multiplier, a 40-bit floating-point adder, and four 40-bit accumulators. The multiplier and the adder work in parallel to perform 25 million floating-point computations per second. The DAU also incorporates special-purpose hardware for data-type conversions.

On-chip memory includes 1536 words of RAM. Up to 16 Mbytes of external memory can be directly addressed by the external memory interface that supports wait states and bus arbitration. All memory can be addressed as 8-, 16-, or 32-bit data, with the 32-bit data being accessed at the same speed as 8- or 16-bit data.

The DSP32C has three I/O ports: an external memory port, a serial port, and a 16-bit parallel port. In addition to providing access to commercially available memory, the external memory interface can be used for memory-mapped I/O. The serial port can interface to a time division multiplexed (TDM) stream, a codec, or another DSP32C. The parallel port provides an interface to an external microprocessor. Three on-chip direct-memory-access (DMA) controllers support direct memory access via the serial input, serial output, and parallel I/O ports.

A single-level interrupt facility can respond to four internal and two external, individually maskable sources. A relocatable vector table controls program flow based on the source of the interrupt.

The DSP32C has two general types of instructions that correspond to the two execution units: data arithmetic (DA) instructions and control arithmetic (CA) instructions. Primarily, DA instructions perform 32-bit floating-point multiply/accumulate operations for signal processing algorithms. Other DA instructions convert the DSP32C internal floating-point data to and from each of the following types: 8-, 16-, or 24-bit 2's complement integer, 8-bit μ-law, 8-bit A-law, or 32-bit single-precision IEEE standard floating-point. The CA instructions perform microprocessor operations such as 16- and 24-bit integer arithmetic and logic functions, conditional branching, and moving data.

In summary, some of the key features of the DSP32C are as follows:

- Full, 32-bit floating-point arithmetic
- Four memory accesses per instruction cycle
- Single, linear address space
- Byte addressable internal and external memory
- Up to 16 Mbytes of external memory
- 1536 words of on-chip RAM
- 16 Mbits/s serial I/O ports and 16-bit parallel I/O port with DMA options
- 8- or 16-bit microprocessor interface requiring no additional logic
- Single-precision, single-cycle, IEEE floating-point conversion capability
- Bit reverse addressing mode
- Four 40-bit accumulators
- Zero-overhead looping
- 16- and 24-bit integer operations
- Vectored Internal and external interrupts with hardware context save
- Low-power CMOS technology

The DSP32C also includes extensive hardware and software support from both AT&T and many third parties. A bus functional simulation model is offered by Logic Automation, a complete DSP32C hardware emulator is available from Hewlett-Packard (64773). For more information on the DSP32C, refer to the DSP32C Information Manual, DSP32C Data Sheet, or any of several software support documentation manuals available from your AT&T account manager.

1.2.3 The AT&T DSP3210 Digital Signal Processor

The DSP3210 Digital Signal Processor is the newest member of AT&T's Floating-Point DSP Family. The DSP3210 is an enhancement of the DSP32C architecture designed specifically to meet the needs of the PC/Workstation multimedia market. The DSP3210 is a three-part product composed of the DSP3210 device, the VCOS real-time OS, and the VCOS multimedia function library. The DSP3210 is a very low-cost systems solution (no local DSP RAM is required for systems solutions) with complete applications support and full, 32-bit floating-point processing power. The DSP3210 is source code compatible with the DSP32C via AT&T's DSP32C to DSP3210 source code translator. New features added to the DSP3210 are briefly outlined in Table 1-1.

Table 1-1 DSP3210 Features

Feature	Benefit
32-bit addressing	Compatible with host address space.
Barrel Shifter	Better bit manipulation.
2048 32-bit words of contiguous on-chip RAM	RAM easily traded between program and data.
Big or little endian byte ordering	Compatible with Motorola and Intel processors.
Compatibility with Motorola and Intel bus protocols (byte select of size/offset)	Simplifies hardware design.
Quad-word transfer capability	Increased system performance.
Programmable 32-bit timer	Reduces system cost.
Interruptible do-loops that can be used in interrupt routines	Improves software performance.
Powerdown mode (wait for interrupt)	Reduces power dissipation.
SIO with DMA transfer counters	Simplifies control of serial DMA channels.
Bit I/O Unit	Simplifies hardware design.
Memory mapped I/O	Expandable architecture.
Boot ROM	Flexible initialization, including EPROM boot routine.
Repeat facility for single instruction do loops (no re-fetching of instruction)	Speeds block move operations.
Page break detect signal for DRAMs	Improves memory system performance.
Support for signed characters and unsigned shorts	Improved compiler performance.
Packaged in a low cost 132 PQFP	Low cost, small footprint.

1.3 Support Software Tools

A variety of tools are available to aid in application program development for the DSP3210. These tools are initially offered under the *UNIX*®, *MS-DOS**, or *Macintosh II*† operating systems. In addition to Assembler/Simulator/Compiler support tools, the DSP3210 includes the AT&T VCOS real-time operating system complete with its multimedia function library as well as powerful debugging and testing tools.

AT&T DSP3210 Support Software Tools

Software tools used to create, test, and debug DSP3210 application programs at the assembly language level are packaged in the DSP3210 Support Software Tools package. An assembler, link editor, simulator, and other utilities are included. The simulator, which has capabilities such as single-stepping, breakpointing, and execution profiling, performs precise simulations of the device.

* *MS-DOS* is a registered trademark of Microsoft.
† *Macintosh II* is a registered trademark of Apple Inc.

AT&T DSP3210 Application Software Library

The DSP3210 Application Software Library is a collection of commonly-used signal processing and 3D graphics functions. Each function is carefully benchmarked to allow applications to easily be built from collections of assembly language functions while meeting real-time processing requirements. Many functions are offered in several implementations, giving the programmer flexibility in balancing speed and memory requirements .

AT&T DSP3210 C Language Compiler

The optimizing C Language compiler for the DSP3210 allows application programs to be written in a general, high-level language. In applications where preliminary program development is performed using the C-language on a minicomputer or supercomputer, the source code can be ported to the DSP3210 with a minimal amount of time and effort. A symbolic debugger and several libraries of commonly used arithmetic and signal processing functions are included with the compiler.

1.4 Other Applicable Documentation

The following documents provide specific information on the DSP3210 and its hardware and software development tools. When designing application hardware and software, it is important to have accurate information. Contact your local account representative for the latest issue of a document.

The *AT&T DSP3210 Digital Signal Processor* Data Sheet contains up-to-date timing requirements and specifications, electrical characteristics, and a summary of the instruction set and device architecture.

The *AT&T DSP3210 Support Software Tools* User Manual describes how to install and use the DSP3210 support software.

The *AT&T DSP3210 Application Software Library* Reference Manual contains information on the use and design of each assembly-language routine included in the DSP3210 Application Software Library.

The *AT&T DSP3210 C-Language Compiler* User Manual describes how to set up and use the DSP3210 C compiler.

The *AT&T DSP3210 C-Language Compiler Library* Reference Manual provides the information necessary to implement the C programming language with the DSP3210.

1.5 Assistance

Assistance is available throughout the life of the product. These services include:

- Technical documentation and product samples
- Information on determining and selecting the appropriate hardware and software

The AT&T DSP Bulletin Board provides the latest and most up-to-date information about AT&T DSP products and application assistance:

1200/2400 baud
7 data bits, 1 stop bit, even parity
215-778-4444

For technical assistance or further information including ordering information and part numbers, please contact the nearest office through the following phone numbers:

Domestic (USA & Canada)

Northeast Region
508-626-2161

Mid-Atlantic Region
215-768-2626

Southeast Region
404-446-4700

North Central Region
612-885-4300

South Central Region
214-869-2040

Southwest Region
602-244-1100

Rocky Mountain Region
303-850-5415

Pacific Northwest
408-522-5555

Southern California
714-220-6223

International

Europe (except Spain & Portugal)
+49 89 950 86 0
Telefax: +49 89 950 86 111

Spain and Portugal
+34 1 404 6012
Telefax: +34 1 404 6252

Japan
813-593-3301
Telex: J32562 ATTIJ
Telefax: 813-593-3307

Pacific Rim
65-225-5233
Telex: RS 42898
Telefax: 65-225-8725

Internal (AT&T Customers)

AT&T internal customers should contact their local AT&T Account Management Office.
If the Account Management telephone number is not known, call 1-800-372-2447 and ask for the telephone number of your account representative.

Chapter 2

Applications

CHAPTER 2. APPLICATIONS

2. APPLICATIONS 2-1
2.1 PC/Workstation Multimedia Applications 2-2
Figure 2-1 Typical PC Multimedia System Configuration 2-2
2.2 3-D Graphics Applications 2-3
Figure 2-2 Typical 3-D Graphics System Configuration 2-3

2. APPLICATIONS

The DSP3210 can be used in many different application areas including: telecommunications, speech processing, image processing, graphics, array processors, robotics, studio electronics, instrumentation, and military applications.

ELECTRONIC DATA PROCESSING

■ Mass Memory	Disc controllers, high-precision servo control
■ Workstations	Graphics, translations, rotations, shading, perspective scaling, inversion, multiplication, numeric accelerators, array processing
■ Front-End Processor	Bit manipulation, encryption

INDUSTRIAL

■ Robotics	High-precision servo control
■ Image Processing	Restoration, pattern recognition, compression
■ Process Control	Minicomputer functions
■ Real-Time Simulators	Graphics, servo control, system modeling
■ Instrumentation	Oscilloscopes, FFT, spectrum analysis, signal generators

TELECOMMUNICATIONS

■ PBX	Tone detection, tone generation, MF, DTMF
■ Switches	Tone detection, tone generation, line testing
■ Modems	Echo cancellation, filtering, error correction and detection
■ Transmission	Multi-pulse LPC, ADPCM, transmultiplexing, encryption

GOVERNMENT/MILITARY

■ Sonar	Beam forming, FFT
■ ECM	FFT, adaptive filtering
■ Airframe	Simulation
■ Radar Tracking	Precision FFT, matrix inversions

SPEECH

■ Recognition	Feature extraction, spectrum analysis, pattern matching
■ Synthesis	LPC, format synthesis
■ Coding	ADPCM, LPC, multi-pulse LPC, vector quantization

CONSUMER

■ Studio Electronics	Digital audio
■ Entertainment	High-end video (special effects)
■ Educational	

When the software required for a DSP application is being developed, it is often desirable to use some standard algorithms already in existence. Each application requires a somewhat different use of these algorithms, yet there are great similarities from the processing point of view. Many of the algorithms available are usable in a wide variety of physical problems. The driving force behind most of the algorithms has been an attempt to reduce the computational and data transfer requirements to accommodate the performance constraints of available hardware. Many of these algorithms have been coded in DSP3210 assembly language and are presented in the *AT&T DSP3210 Application Software Library* Reference Manual.

2.1 PC/Workstation Multimedia Applications

The DSP3210 is intended to be used in PC and workstation system architectures in which the DSP3210 is a parallel processor to a host processor. The DSP3210 maintains a 32-bit bus master interface to system memory (see Figure 2-1). The primary benefit of this system architecture is that the DSP is able to access program and data from system memory without host intervention, and expensive local SRAM is replaced by general purpose DRAM. Since the DSP3210 supports both big- and little-endian byte ordering, sharing both data and pointer values with any host microprocessor is easily accomplished. This is especially useful in multimedia applications where intimate communications between the host µP and DSP are necessary. For real-time signal processing, on-chip SRAM is loaded with code and data from system memory before executing. Typically, applications are broken down into functions that are executed successively in this fashion.

This technique is also used to create low-cost EISA and MCA add-on cards. The DSP3210 acts as a bus master on the 32-bit bus to access system memory (rather than adding memory to the DSP3210 card). ISA 16-bit add-on cards typically employ local memory since their 16-bit width would present a memory bottleneck to the DSP3210. However, the visible caching technique employed by the VCOS operating system permits the use of slow, inexpensive DRAM for local memory with minimal impact on system performance. Applications are compatible across all implementations described here.

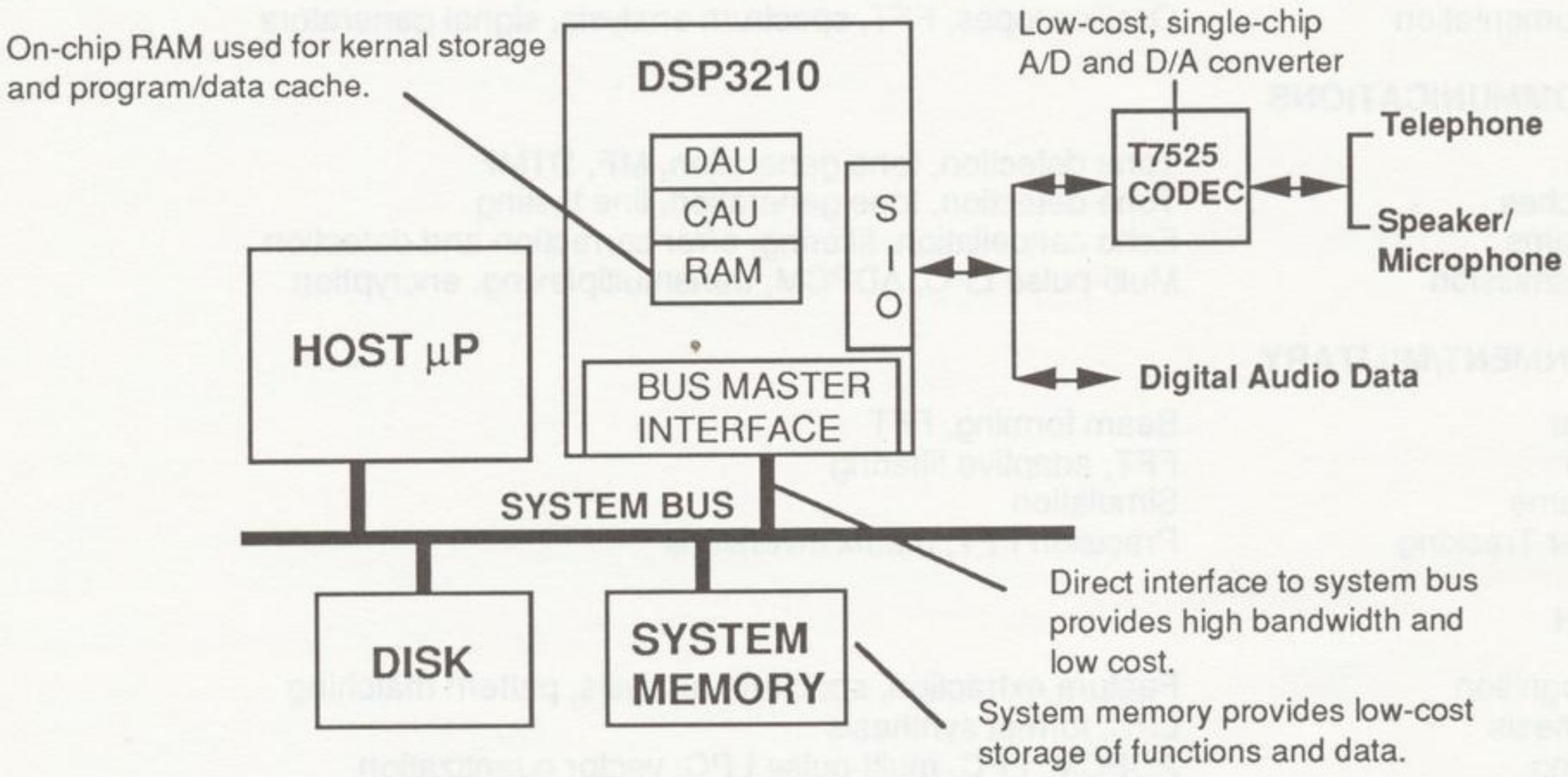


Figure 2-1 Typical PC Multimedia System Configuration

2.2 3-D Graphics Applications

Although the DSP3210 was primarily designed for personal computer/workstation multimedia applications, it is ideally suited to perform the floating-point, compute intensive calculations in a typical 3-D transformation pipeline. In fact, while performing graphics operations, the DSP3210 often outperforms far more costly processors which are touted for their graphics capabilities. These operations consist of matrix multiplies, divides, square root computations, comparisons, and integer conversions. In addition to its native floating-point performance, byte addressability, and large address space, the DSP3210 has on-chip hardware to improve the efficiency of many of these operations such as: reciprocal seed, 8-, 16-, and 32-bit integer to/from floating-point conversion, clip-test register, Z-buffering, etc. A typical graphics system will contain multiple DSP3210 processors that operate in parallel to render the image. The low cost of the DSP3210 makes multiple DSP graphics systems practical. Its page-break and quarter-cycle wait state features significantly reduce system costs while increasing system performance. Inexpensive DRAM can be used for local memory while still achieving the high throughput necessary for high-speed 3-D transformations. The DSP3210's flexible bus protocol also makes it possible for several DSP3210 devices to share the same command and output buffers with little or no performance degradation, further reducing system costs. Figure 2-2 shows a typical 3-D graphics subsystem architecture employing a common command buffer/output buffer configuration.

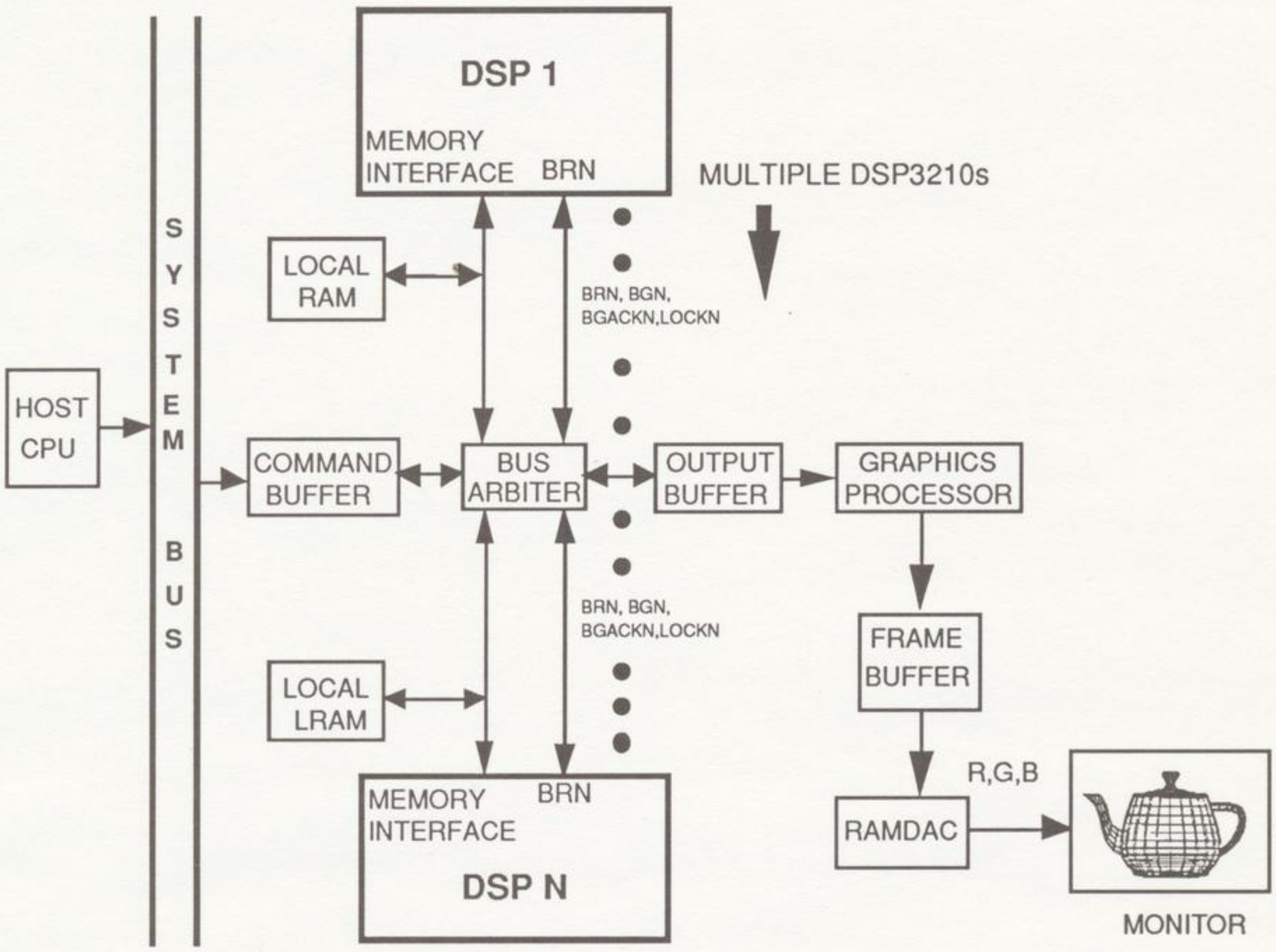


Figure 2-2 Typical 3-D Graphics System Configuration

Chapter 3

DSP3210 Architecture

CHAPTER 3. DSP3210 ARCHITECTURE

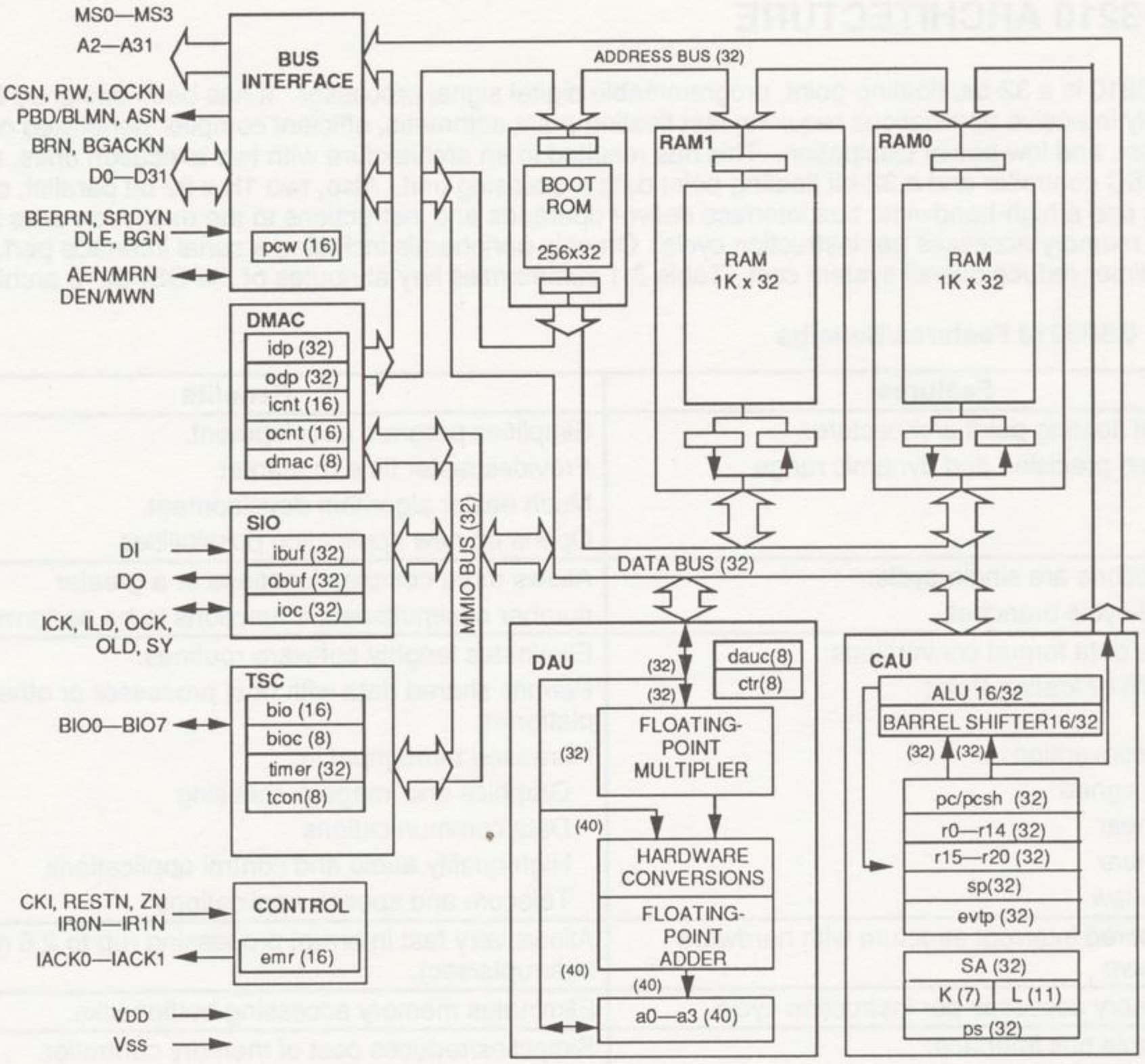
3. DSP3210 ARCHITECTURE 3-1
Table 3-1 DSP3210 Features/Benefits 3-1
Figure 3-1 DSP3210 Block Diagram 3-2
3.1 Functional Units 3-3
3.1.1 Control Arithmetic Unit (CAU) 3-3
3.1.2 Data Arithmetic Unit (DAU) 3-3
3.1.3 On-Chip Memory 3-3
3.1.4 Bus Interface 3-4
3.1.5 Serial I/O (SIO) 3-4
3.1.6 DMA Controller (DMAC) 3-4
3.1.7 Timer 3-5
3.1.8 Bit I/O (BIO) 3-5
3.2 Processor Control Features 3-5
3.2.1 Serial I/O DMA 3-5
3.2.2 Exception Processing 3-5
3.2.3 Wait-for-Interrupt 3-6
3.3 Data Types 3-6
3.3.1 32-bit 2's Complement Data Type 3-6
3.3.2 16-bit 2's Complement Data Type 3-7
3.3.3 32-bit Floating-Point Data Type 3-7
Figure 3-2 DSP32C Internal 32-bit Floating-Point Format 3-7
3.4 Memory Organization 3-8
Figure 3-3 Memory Addressing 3-9
Figure 3-4 Memory Map 3-10
Figure 3-5 IO Memory Map 3-11
3.5 Addressing Modes 3-11
Table 3-2 Addressing Modes Allowed in Each Instruction Type 3-12
3.5.1 Short Immediate 3-12
3.5.2 24-bit Immediate 3-12
3.5.3 CAU Register (rN) Direct 3-12
3.5.4 IO Register (iorN) Direct 3-12
3.5.5 DAU Register (aN) Direct 3-12
3.5.6 Memory Direct 3-12
3.5.7 Register Indirect 3-13
3.5.8 Register Indirect with Postmodification 3-13
3.5.9 Other Addressing Modes 3-13
3.6 IO Registers 3-14

3. DSP3210 ARCHITECTURE

The DSP3210 is a 32-bit, floating-point, programmable digital signal processor. It has been designed for use in numerically intensive applications requiring fast floating-point arithmetic, efficient compiler-generated code, low system cost, and low power dissipation. This has resulted in an architecture with two execution units, a 32-bit integer RISC controller and a 32-bit floating-point data processing unit. Also, two 1K x 32 bit parallel, on-chip memories and a high-bandwidth bus interface deliver operands and instructions to the execution units supporting up to four memory accesses per instruction cycle. On-chip peripherals including a serial interface port, bit I/O port, and timer reduce overall system cost. Table 3-1 summarizes key attributes of the DSP3210 architecture.

Table 3-1 DSP3210 Features/Benefits

Features	Benefits
Full 32-bit floating-point architecture: –Increased precision and dynamic range	Simplifies program development. Provides faster time to market. Much easier algorithm development. Opens up new application possibilities.
All instructions are single-cycle: –No multi-cycle branches	Allows more complex functions or a greater number of simultaneous functions to be performed.
Hardware data format conversions: –IEEE P754 Floating-Point –Integer conversion 8-bit unsigned 16-bit linear 32-bit linear µ-law/A-law	Eliminates lenghty software routines. Permits shared data with host processor or other platforms. Increased throughput in: Graphics and image processing Data communications High quality audio and control applications Telecom and speech applications
Fully vectored interrupt structure with hardware context save	Allows very fast interrupt processing (up to 2.6 million interrupts/sec).
Four memory accesses per instruction cycle	Eliminates memory accessing bottlenecks.
Configurable bus interface: –32-bit addressing –Support for relinquish/retry/bus error –Page break detect	Simplifies/reduces cost of memory controller. Compatible with µprocessor system busses. Improves performance in designs using DRAM.
Byte-addressable address space	Efficient storage of 8- and 16-bit data. Address pointer compatibility with microprocessors.
Flexible wait-state facility: –Each wait-state is 1/4 instruction cycle –Two independently configurable external memory speed partitions	Greater memory speed selection flexibility than conventional full cycle wait-states. Allows mixing of slow and fast memory. Optimizes system speed/cost requirements.
On-chip peripherals: –Serial I/O with DMA capability –32-bit timer	Low cost interface to external devices. Lowers system cost.



LEGEND:

CAU	Control Arithmetic Unit	SIO	Serial Input/Output
DAU	Data Arithmetic Unit	TSC	Timer/Status/Control
RAM	Random Access Memory	DMAC	DMA Controller
ROM	Read Only Memory	MMIO	Memory Mapped I/O

Figure 3-1 DSP3210 Block Diagram

3.1 Functional Units

The DSP3210 consists of seven functional units: Control Arithmetic Unit (CAU), Data Arithmetic Unit (DAU), On-chip Memory (RAM0, RAM1, Boot ROM), Bus Interface, Serial I/O (SIO), DMA Controller (DMAC), and Timer/Status/Control (TSC). Figure 3-1 is a block diagram of the DSP3210.

3.1.1 Control Arithmetic Unit (CAU)

The CAU is responsible for performing address calculations, branching control, and 16- or 32-bit integer arithmetic and logic operations. It is a RISC core consisting of a 32-bit arithmetic logic unit (ALU) that performs the integer arithmetic and logical operations, a 32-bit program counter (PC), and twenty-two 32-bit general-purpose registers. The CAU can execute 16.7 MIPS (Million Instructions Per Second).

The CAU performs two tasks: one executes integer, data move, and control instructions (CA instructions), and the other generates addresses for the operands of floating-point instructions (DA instructions). CA instructions perform load/store, branching control, and 16- and 32-bit integer arithmetic and logical operations. DA instructions can have up to four memory accesses per instruction, and the CAU is responsible for generating these addresses using the post-modified, register-indirect addressing mode—one address in each of the four states of an instruction cycle. For a more detailed description of the CAU, see Section 8.1 Control Arithmetic Unit.

3.1.2 Data Arithmetic Unit (DAU)

The DAU is the primary execution unit for signal processing algorithms. This unit contains a 32-bit floating-point multiplier, a 40-bit floating-point adder, four 40-bit accumulators, and two control registers (dauc and ctr). The multiplier and adder work in parallel to perform 16.7 million computations per second of the form (a=b+c*d). The multiplier and adder each produce one result per instruction cycle. The DAU contains a four stage pipeline: operand load, multiply, accumulate, and result store. Thus, in any instruction cycle, the DAU may be processing four different instructions, each in a different stage of execution.

The DAU supports two floating-point formats, single precision (32-bit) and extended single precision (40-bit). Extended single precision provides 8 additional mantissa guard bits. Post-normalization logic transparently shifts binary points and adjusts exponents to prevent inaccurate rounding of bits when the floating-point numbers are added or multiplied, eliminating concerns like scaling and quantization error. All normalization is done automatically, so the result in the accumulator is fully normalized.

Single instruction, data type conversions are done in hardware in the DAU, reducing overhead required to do these conversions. The DAU performs data type conversions between the DSP3210 32-bit floating-point format and IEEE P754 standard 32-bit floating-point, 16- and 32-bit integer, 8-bit unsigned, u-law and A-law formats. The DAU also provides an instruction to convert a 32-bit floating-point operand to a 3-bit seed value used for reciprocal approximation in division operations. For a more detailed description of the DAU, see Section 8.2 - Data Arithmetic Unit.

3.1.3 On-Chip Memory

The DSP3210 provides on-chip memory for instructions and data. Instructions and data can arbitrarily reside in any location in the on-chip memory. The DSP3210 provides two 1K x 32 RAMs and a 256 x 32 Boot ROM. The memories operate in a parallel fashion to achieve high performance. The Boot ROM is preprogrammed with a routine that can load the internal RAM of the DSP from an external memory, such as an EPROM.

3.1.4 Bus Interface

The external address bus of the DSP3210 is 32-bits wide and fully byte-addressable, allowing the DSP3210 to directly address 4 Gbytes of memory or memory-mapped hardware. External memory is partitioned into two logical address spaces A and B. Each partition contains approximately 2 Gbytes of address space. The number of wait-states for external memory paritions A and B are independently configurable via the pcw register. Configured waits of 0-, 1-, 2-, or 3-or-more wait-states are programmable; this simplifies the interface to fast external memory. Unlike most digital signal processors (which employ full-cycle wait-states), the DSP3210 offers much greater flexibility by offering 1/4 cycle wait-states. Each wait-state is 1/4 of an instruction cycle, allowing much greater granularity in determining optimal speed/cost memory trade-offs. When waits are externally controlled, the DSP adds wait-states until the memory acknowledges the transaction via the SRDYN pin. The bus interface supports retry, relinquish/retry, and bus error exception handling. All signalling provided to the external system is configurable on reset to simplify the interface to a variety of microprocessor system buses.

Sharing the external memory interface is performed via a complete request/acknowledge protocol. System throughput is greatly enhanced by the DSP3210's ability to execute from internal memory while the DSP3210 does not have ownership of the bus. The DSP3210 continues to execute from internal memory until accesses to the external memory are needed. At that point, the DSP asserts the BRN signal and wiats for BGN. The bus arbiter acknowledges the bus request by asserting the DSP3210's bus grant pin, BGN. The DSP3210 then acknowledges the grant by asserting the bus grant acknowledge, BGACKN, and drives the external memory interface pins. When BGN is negated, any on-going external memory transaction is completed before the DSP relinquishes the bus by placing the external memory interface bus in the high-impedance state and negating BGACKN. See Section 6 - Bus Interface.

The DSP3210 shares the external memory interface bus through a request/acknowledge protocol. The external memory interface bus consists of the address bus, A2—A31, the data bus, D0—D31, and associated control pins (See Section 6 - Bus Interface). The DSP is a default bus slave. When the DSP does not have ownership of the bus, it executes from internal memory until accesses to the external memory are needed, and at that point the DSP asserts the BRN and waits for BGN. The bus arbiter acknowledges the bus request by asserting the DSP's bus grant pin, BGN. The DSP acknowledges the grant by asserting the bus grant acknowledge, BGACKN and driving the external memory interface pins. When the BGN is negated, any on-going external memory transaction is completed before the DSP relinquishes the bus by placing the external memory interface bus in the high-impedance state and negating BGACKN. See Section 6 Bus Interface.

3.1.5 Serial I/O (SIO)

The SIO unit provides serial communications and synchronization with external devices. The external signals provided support a direct interface to a time-division-multiplexed (TDM) line, a zero-chip interface to codecs, and direct DSP-to-DSP transfers for multiprocessor applications. The SIO performs serial-to-parallel conversion of input data and parallel-to-serial conversion of output data, at a maximum rate of 25 Mbits/s. It is composed of a serial input port, a serial output port, and on-chip clock generators. Both ports are double buffered so that back-to-back transfers are possible. The SIO is configurable via the ioc register. The input buffer, ibuf, the output buffer, obuf, and the ioc register are accessible as MMIO (Memory-mapped Input/Output) registers in the instruction set.

The data sizes of the serial input and output can be selected independently. Input data lengths of 8, 16, and 32 bits, and output data lengths of 8, 16, 24, and 32 bits can be selected. The input and output data may be selected to be most significant bit first or least significant bit first independently.

SIO transfers can be made under program, interrupt, or DMA control. A program may test the input or output buffer status flags using conditional branch instructions. By configuring the exception mask register, emr, interrupt requests may be generated by the input and output buffer status flags. In DMA mode, transfers occur between ibuf, obuf, and memory without program intervention.

For a detailed description, see Section 9.1 Serial I/O.

3.1.6 DMA Controller (DMAC)

The DMA controller contains two DMA channels, one for input DMA and one for output DMA , that are used in conjunction with the serial I/O. By configuring the input DMA channel, data being shifted into the serial input port

can be buffered in memory without processor intervention. By configuring the output DMA channel, a buffer of data in memory can be supplied to the serial output, as necessary, without processor intervention. The registers to configure the DMA Controller are accessible as MMIO registers in the instruction set. By configuring the exception mask register, emr, interrupt requests may be generated when the memory buffer has been filled or emptied based on the size of the buffer requested.

For a detailed description see Section 9.2 DMA Controller.

3.1.7 Timer

The timer is a programmable 32-bit interval timer/counter that can be used for interval timing, rate generation, event counting, or waveform generation. The input to the timer can be derived from the DSP3210 clock, or it may come from an external source. The output of the timer can generate a maskable interrupt or be selected as an output of the chip to drive external hardware. The count-down timer can be configured to count to zero once, or to count continuously by automatically reloading the counter with its initial value when it reaches zero. The count value may be read or changed at any time during operation. The registers associated with the timer are accessible as MMIO registers in the instruction set. By configuring the exception mask register, emr, interrupt requests may be generated when the count reaches zero.

For a detailed description, see Section 9.3 Timer.

3.1.8 Bit I/O (BIO)

The BIO is a general purpose 8-bit input/output port. It includes features that make it suitable for board-level status signal generation and control signal testing by the DSP3210. The BIO interface consists of 8 I/O lines, which can be independently configured as an input or an output. Outputs can be written with a 1 or a 0, toggled, or left unchanged. Inputs can be directly read and loaded into a CAU register and then tested. The registers associated with the BIO are accessible as MMIO registers in the instruction set. Four of the BIO ports are used on reset to configure the memory map and bus interface. After reset, they can be used arbitrarily.

For a detailed description, see Section 9.4 Bit I/O.

3.2 Processor Control Features

The DSP3210 supports advanced control features that simplify system design and improve software performance. This section overviews serial I/O direct-memory access (DMA), exceptions, and the powerdown mode. A more detailed description of these control functions is presented in Section 7 Processor States.

3.2.1 Serial I/O Direct Memory Access (DMA)

External devices can access the on-chip RAM in the DSP3210, as well as external memory, using DMA. DMA transfers occur between the memory and the serial I/O ports without processor intervention, using cycle-stealing. Two on-chip DMA controllers support memory access via the serial input and serial output ports. See Section 7.1 Serial I/O and Section 7.2 DMA Controller.

3.2.2 Exception Processing

Normal instruction processing can be altered by the introduction of interrupt routines or error handling routines. Exception processing is the set of activities performed by the processor in preparing to execute a handler routine or in returning to the program that took the exception. Error exception and interrupt exceptions cause different activities to be performed. In particular, error exceptions abort the current instruction and can not resume processing. Interrupt exceptions shadow the current state of the processor before taking the interrupt exception; therefore, when the interrupt routine is complete the program can be reinstated and continued.

The error and interrupt exceptions are prioritized and some error sources in addition to all interrupt sources are individually maskable via the exception mask register, emr. A relocatable vector table controls program flow based on the source of the interrupt exception. In response to a given exception, the DSP branches to the corresponding address in the exception vector table which contains pairs of 32-bit words.

The DSP3210 provides a zero-overhead context save of the entire DAU (floating-point unit) for interrupt processing. Interrupt latency is only three instruction cycles for interrupt entry and one instruction cycle for an interrupt exit. Before servicing the interrupt, the DSP3210 automatically saves the state of the machine that is invisible to the programmer, as well as the DAU accumulators a0—a3 (including guards bits), all DAU flag status information, and the dauc register. Internal states that are visible to the programmer are saved and restored by the interrupt service routine. To return to the interrupted program, the interrupt service routine restores the user-visible state of the DSP3210 (that was saved) and then executes the **ireturn** instruction. This single-cycle interrupt return automatically restores the entire DAU state saved during interrupt entry. Quick interrupt entry/exit/context save is critical to real-time multimedia tasks.

For a detailed description of exception processing, see Section 7.4 Instruction Sequencing Exceptions.

3.2.3 Wait-for-Interrupt

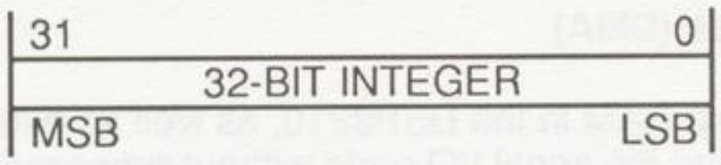
The DSP3210 is equipped with a wait-for-interrupt instruction that stops internal execution in the DAU and CAU. External memory and internal memory operations execute to completion and then wait. Bus arbitration logic remains active in this mode. The peripheral units (Serial I/O, DMA controller, Timer, and BIO) remain active during wait-for-interrupt to perform I/O events and the interrupt handler is active to sense an interrupt request. The DSP will exit the wait-for-interrupt mode when either an unmasked interrupt is requested or an error exception occurs.

3.3 Data Types

Internally, the DSP3210 fully supports three data type formats: 16- and 32-bit 2's complement integers and 32-bit, floating-point numbers. The control arithmetic unit (CAU) performs arithmetic and logical operations on the 16- and 32-bit integer formats. It also has the ability to load and store 8-bit signed and unsigned, and 16-bit unsigned data. The data arithmetic unit (DAU) performs multiply and accumulate operations on the 32-bit, floating-point data and local to the DAU supports a 40-bit extended precision format. The DAU also performs data type conversion between IEEE P754 single precision, 8-bit unsigned, μ-law and A-law, and 16- and 32-bit 2's complement.

3.3.1 32-bit 2's Complement Data Type

The CAU supports a 32-bit, 2's complement integer data type that can be used as 32-bit integer data or for memory pointers. The 32-bit integers range in value from -2^{31} to $2^{31} - 1$. Thirty-two bit arithmetic operations are the default type. All pointers must be 32-bit data types. The organization of 32-bit data in CAU registers is:



3.3.2 16-bit 2's Complement Data Type

The CAU supports a 16-bit, 2's complement integer data type that is used as 16-bit integer data. The 16-bit integers range in value from -2^{15} to $2^{15} - 1$. Because CAU registers are 32-bits wide, when a 16-bit integer operation is performed, data should be loaded into the lower 16-bits of the register (with a sign-extend of bit 15 into the upper 16 bits). A 16-bit operation performs 32-bit arithmetic with flags computed according to 16-bit operation flag rules (see Section 4.1 Flags). If the results are stored back to memory, the lower 16-bits of the result should be stored into a 16-bit memory location. The organization of 16-bit data in CAU registers is:

31 16	15 0
SIGN-EXTENDED	16-BIT INTEGER
	MSB LSB

3.3.3 32-bit Floating-Point Data Type

There are two floating-point formats used inside the DAU: a 32-bit format and a 40-bit format (32-bits, plus 8 mantissa guard bits). The latter format, used only inside the DAU, is not supported by the rest of the device. The DAU accumulators are 40-bits. The guard bits are not user-accessible. For more information about the 40-bit format, see Section 8.2 - Data Aritmetic Unit. The organization of floating-point data in DAU accumulators (a0—a3) is:

39	38 16	15 8	7 0
s	f	g	
	FRACTION	GUARD	EXPONENT
s	MSB LSB	MSB	MSB LSB

The following is a description of the 32-bit floating-point format.

Format

The DSP3210 internal 32-bit floating-point format consists of a 24-bit, normalized, 2's complement mantissa and an 8-bit biased exponent. Figure 3-2 shows the DSP3210 32-bit floating-point format.

31	30 8	7 0
s	f	e
s	FRACTION	EXPONENT
	MSB LSB	MSB LSB

Figure 3-2 DSP3210 Internal 32-bit Floating-Point Format

where: s = the sign bit (0 for positive, 1 for negative)
f = fractional part of the mantissa
s,f = the mantissa
e = the exponent

This floating-point quantity is represented in decimal as follows:

$$N = M * 2^{(e-128)}$$

where: N = the decimal value
e = the exponent, the unsigned quantity "eeeeeeee" expressed in base 10.
M = mantissa, the 2's complement quantity "s(!s).ffff...ffff is equivalent to $(-2)^s$ + 0.ffff...ffff expressed in base 10.
* = multiplication

The magnitude of the mantissa is always normalized to lie between 1 and 2. So, the leading 1 (or leading 0 if a negative mantissa) does not appear explicitly in the floating-point word. This implicit bit is the s bar (!s) that appears above.

Zero

A floating-point value with e=0 is reserved to represent the number zero. Here, the sign and mantissa bits must also be zero. A zero exponent with a nonzero sign and/or mantissa is called a dirty zero, but is treated the same as zero. If a dirty zero is an operand, it is treated as a zero. Any dirty zero generated in the floating-point multiplier or adder is flushed to a valid zero before being written to the accumulators or memory.

Range

The 32-bit floating-point format used in the DSP3210 provides over 1500 dB of dynamic range. The range of nonzero positive floating-point numbers is:

$N = [1*2^{-127}, [2-2^{-23}] * 2^{127}]$ inclusive.

The range of nonzero negative floating-point numbers is:

$N = [-[1+2^{-23}] * 2^{-127}, -2 * 2^{127}]$ inclusive.

The ranges of positive and negative nonzero numbers are almost identical, and the corresponding range for the magnitude of a floating-point number N, in decimal, is approximately:

|N| = [5.87747E-39, 3.40282E38]

3.4 Memory Organization

The memory space is byte addressable and can support both little endian and big endian byte orderings. The organization of bytes in a 32-bit word is based on the state of the B/LN bit in the pcw register. If B/LN = 0, the little endian mode is selected and byte 0 is the least significant byte of the 32-bit word. If B/LN = 1, the big endian mode is selected and byte 0 is the most significant byte of the 32-bit word. 8-bit data is referred to as a byte, 16-bit data is referred to as a half-word, and 32-bit data is referred to as a word. Each 32-bit word is organized as four bytes, two half-words, or one word. Data must be aligned on the same byte boundary as its operand size. Bytes can be arbitrarily aligned, half words must be aligned on bytes 0 or 2, and words must be aligned on byte 0. Figure 3-3 illustrates the addressing of data for both byte orderings.

Note: $ indicates address

Big Endian (B/LN=1)			
31 23 15 7 0			
Word $00000000			
Half-word $00000000		Half-word $00000002	
Byte $00000000	Byte $00000001	Byte $00000002	Byte $00000003
Word $00000004			
Half-word $00000004		Half-word $00000006	
Byte $00000004	Byte $00000005	Byte $00000006	Byte $00000007
etc.			

Little Endian (B/LN=0)			
31 23 15 7 0			
Word $00000000			
Half-word $00000002		Half-word $00000000	
Byte $00000003	Byte $00000002	Byte $00000001	Byte $00000000
Word $00000004			
Half-word $00000006		Half-word $00000004	
Byte $00000007	Byte $00000006	Byte $00000005	Byte $00000004
etc.			

Figure 3-3 Memory Addressing

The organization of memory in the DSP3210 address space is shown in Figure 3-4. One of the two memory maps is selectable with the C/PN bit of the pcw register. The memory map is divided into seven major sections: Boot ROM, MMIO (Memory-mapped Input/Output), on-chip RAM0, on-chip RAM1, Reserved (for future expansion of ROM and RAM), External Memory A and External Memory B. On-chip SRAM resources consist of two 1K x 32 banks. The Boot ROM consists of 256 words. The MMIO reserves 256 words. The reserved expansion region provides for future expansion of up to 15.5K words of on-chip memory. External Memory A and External Memory B are logical partitions in which a different number of wait-states may be configured for each. The number of wait-states is configured in the pcw register. See Section 6 Bus Interface.

The first address issued on reset is location 0x0 and is located in Boot ROM (if C/PN is initialized to a 1 on reset by BIO7) or external memory (if C/PN is initialized to a 0). The generic code provided in the ROM will allow redirection of the starting address to locations in external memory A or B. Also, the capability to load internal RAM from an external memory device, such as EPROM is provided. This simplifies the use of the DSP3210 in embedded applications. The ROM code is described in Appendix A.

The MMIO memory space reserves 256 words for accessing on-chip memory-mapped IO registers. The DSP3210 contains three peripheral units: SIO, TSC, and DMAC. The control and data registers for these peripherals are addressable in the memory space of the DSP3210 as shown in Figure 3-5 IO Memory Map. Each register occupies one word memory location. For registers smaller than 32-bits, Figure 3-5 shows the location of valid data in the register. MMIO registers are a fixed size and must be read and written using the corresponding size indicator. The address of the MMIO register is dependent on the byte ordering.

Note: Some IO registers have reserved bits for future definition. Those particular bits must be written as zeros for future compatibility.

Reset Address

0x00000000 0x000003FF	Boot ROM
0x00000400 0x000007FF	MMIO
0x00000800 0x0000DFFF	Reserved
0x0000E000 0x0000EFFF	RAM1
0x0000F000 0x0000FFFF	RAM0
0x00010000 0x5FFFFFFF	External Memory A
0x60000000 0xFFFFFFFF	External Memory B

C/PN = 1

Reset Address

0x00000000 0x5002FFFF	External Memory A
0x50030000 0x500303FF	Boot ROM
0x50030400 0x500307FF	MMIO
0x50030800 0x5003DFFF	Reserved
0x5003E000 0x5003EFFF	RAM1
0x5003F0000 0x5FFFFFFF	RAM0
0x50040000 0x5FFFFFFF	External Memory A
0x60000000 0xFFFFFFFF	External Memory B

C/PN = 0

Figure 3-4 Memory Map

16 LSBs of Address (Big Endian)	16 LSBs of Address (Little Endian)	Symbolic Name	Register Bits 31 23 15 7 0	
0x0400	0x0400	Reserved		
0x0404	0x0404	ibuf		R
0x0408	0x0408	obuf		W
0x040C	0x040C	ioc		R/W
0x0413	0x0410	tcon		R/W
0x0414	0x0414	timer		R/W
0x041B	0x0418	bioc		R/W
0x041E	0x041C	bio		R/W
0x0420	0x0420	idp		R/W
0x0424	0x0424	odp		R/W
0x042A	0x0428	icnt		R/W
0x042E	0x042C	ocnt		R/W
0x0433	0x0430	dmac		R/W
0x0434 0x07FC	0x0434 0x07FC	Reserved		

Reserved Bits R/W : Read/Write W : Write Only

Figure 3-5 IO Memory Map

3.5 Addressing Modes

The DSP3210 supports several data access or addressing modes including immediate, memory-direct, register-direct, register-indirect, and register-indirect with postmodification. The addressing modes allowed for each type of instruction are shown in Table 3-2. CA data move instructions, which are used to load CAU registers, support the greatest number of addressing modes and therefore have the most flexibility. The remainder of the instruction types support fewer addressing modes are more tailored to the intended use of the instruction. See Section 4 Instruction Set for more information about each instruction. This section describes the DSP3210 addressing modes.

Table 3-2 Addressing Modes Allowed in Each Instruction Type

Addressing Mode	Instruction Type			
	CA Data Move Group (CAU Reg)	CA Data Move Group (I/O Reg)	CA Arithmetic/ Logic Group	DA M/A and Special Func.
Short Immediate	Yes			
24-bit Immediate	Yes			
Memory Direct	Yes			
CAU Register Direct	Yes	Yes	Yes	
IO Register Direct	Yes			
DAU Register Direct				Yes
Register Indirect	Yes	Yes		Yes
Register Indirect with Postmodification	Yes	Yes		Yes

3.5.1 Short Immediate

Notation: N (16-bit)

In the short immediate addressing mode, the value **N**, a 16-bit 2's complement integer, is supplied by the instruction, and **N** is MSB-extended to 32-bits.

3.5.2 24-bit Immediate

Notation: M (24-bit)

In the 24-bit immediate addressing mode, the value **M**, a 24-bit unsigned integer, is supplied by the instruction, and **M** is zero-extended to 32-bits.

3.5.3 CAU Register (rN) Direct

Notation: rN (N=0-22)

In the CAU register direct mode, the operand is a CAU register specified by the instruction.

3.5.4 IO Register (iorN) Direct

Notation: iorN (N=1-15)

In the I/O register direct mode, the operand is an IO register specified by the instruction.

3.5.5 DAU Register (aN) Direct

Notation: aN (N=0-3)

In the DAU register direct mode, the operand is a DAU register specified by the instruction.

3.5.6 Memory Direct

Notation: *L (16-bit)

In the memory direct addressing mode, the value **L**, a 16-bit unsigned integer, is supplied by the instruction. The effective address is formed such that it always points to the 64K bytes of on-chip memory space. Therefore, when C/PN =1, the upper sixteen bits of the effective address are 0x0000, and when C/PN=1, the upper 16 bits of the effective address are 0x5003.

3.5.7 Register Indirect

Notation: *rP (P=0-22 for CA instructions,
P=1-14 for DA instructions)

In the register indirect addressing mode, the operand is in memory and the address of the operand is in the register specified by the instruction. This register is referred to as a pointer register. The set of registers that can be used for this addressing mode is dependent on the instruction type. For all CA Data Move instructions, CAU registers r0—r22 can be used as pointer registers. For all DA instructions, CAU registers r1—r14 can be used as pointer registers.

3.5.8 Register Indirect with Postmodification

Notation:

*rP++	post-increment
*rP--	post-decrement
*rP++rI	post-modify by value in register rI
	(P=0—22 and I=0—22 for CA instructions, P=1—14 and I=15—19 for DA instructions)

In the register indirect addressing mode with postmodification, the operand is in memory and the address of the operand is in the register specified by the instruction, rP. This register is referred to as a pointer register. After the operand address is used, it is modified by ±1, ±2, ±4, or the value in register rI. rI is referred to as an increment register. The post-increment mode modifies the pointer register by +1, +2, or +4 depending on the operand size: byte, half-word, or word. The post-decrement mode modifies the pointer register by –1, –2, or –4 depending on the operand size: byte, half-word, or word. The post-modify by register mode modifies the pointer register by the value in the increment register rI. rI is a 2's complement 32-bit integer. The set of registers, pointers, and increments that can be used for this addressing mode is dependent on the instruction type. For all CA Data Move instructions, CAU registers r0—r22 can be used as pointer registers or increment registers. For all DA instructions, CAU registers r1—r14 can be used as pointer registers, and CAU registers r15—19 can be used as increment registers. If the pointer register is the stack pointer, the operand size is always long word and the post-increment and post-decrement is always by 4 to keep the stack pointer aligned to a word boundary.

3.5.9 Other Addressing Modes

Additional addressing modes, including 32-bit immediate, indexed, relative, memory indirect, bit-reversed, and modulo modes can easily be derived by combining two or more CAU instructions as follows.

—The 32-bit immediate addressing mode is achieved by using the two instruction sequences:

rD = N,
rD = rS <<|N.

—The indexed or base plus displacement mode is achieved by using the instruction **rD = rS + N** and then performing register indirect addressing using rD.

—The relative modes are achieved by using the program counter in the instruction **rD = pc + rS** and then performing register-indirect addressing using rD. rS contains the offset from the pc.

—The memory indirect mode is achieved using the instruction **rD = MEM** and then performing register indirect addressing using rD.

—The bit-reversed addressing mode is achieved by using the carry-reverse add function, **rD = rD # rS**, and then performing register indirect addressing using rD.

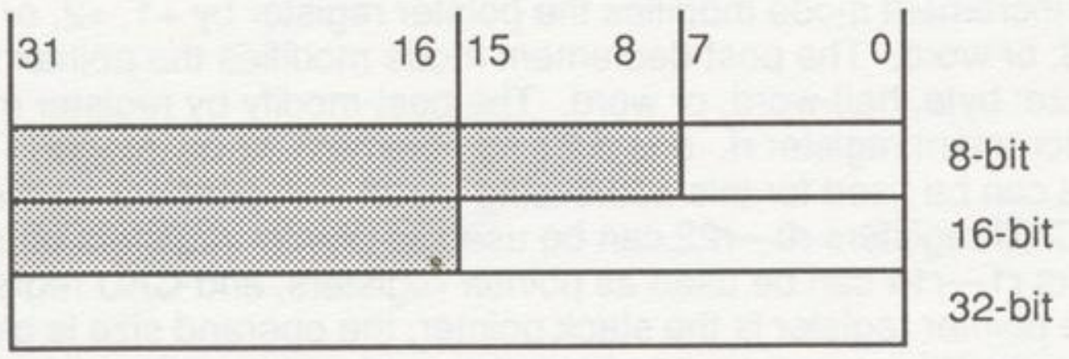
—The modulo addressing mode is achieved by using a two instruction sequence following a postmodification by an increment value. Addresses for the beginning and end locations of the circular buffer are placed in registers.

With:
r1=start_address
r2=end_address
r3=current pointer
r15=increment (step) value
r4=temporary register

Perform postmodification on r3 as usual: **a0=*r3++r15**
Two instruction cycles for modulus operation: **r4=r3-r2**
if(pl) r3=r1+r4

3.6 IO Registers

The IO register set includes control registers and data registers. IO registers vary in size according to their function. IO registers can be 8-bit, 16-bit, or 32-bit registers. The organization of data in the three sizes of registers is:



In general, IO registers are a fixed size and must be read and written using the corresponding size indicator. The IO register set consists of:

IO#	Name	Size
ior0	ps	16-bit
ior8	emr	16-bit
ior12	pcw	16-bit
ior14	dauc	8-bit
ior15	ctr	8-bit

Note: Some IO registers have reserved bits for future definition. Those particular bits must be written as zeros for future compatibility.

Chapter 4

Instruction Set

CHAPTER 4. INSTRUCTION SET

4 INSTRUCTION SET 4-1
Figure 4-1 Register Programming Model 4-2
4.1 Flags 4-3
Table 4-1 DSP3210 Flags 4-3
4.2 DA Instructions 4-4
4.2.1 DA Instructions - Multiply/Accumulate 4-4
Table 4-2 DA Multiply/Accumulate Instructions 4-4
Table 4-3 Replacement for DA Multiply/Accumulate Instructions 4-5
4.2.2 DA Instructions - Special Functions 4-5
Table 4-4 DA Special Function Instructions 4-5
Table 4-5 Replacement for DA Special Function Instructions 4-5
4.3 CA Instructions 4-6
4.3.1 CA Instructions - Control 4-6
Table 4-6 CA Control Instructions 4-6
Table 4-7 Replacement for COND 4-7
4.3.2 CA Instructions - Arithmetic/Logic 4-8
Table 4-8 CA Arithmetic/Logic Instructions 4-9
4.3.3 CA Instructions - Data Move 4-10
Table 4-9 CA Data Move Group Instructions 4-11
Table 4-10 Replacement for CA Data Move Instructions 4-11
4.4 DSP3210 Programming 4-12
4.4.1 Restrictions 4-12
4.4.1.1 Restriction 1 - Z-Field Pointer 4-12
4.4.1.2 Restriction 2 - CAU and IO Register Store 4-12
4.4.1.3 Restriction 3 - CAU Register Load 4-13
4.4.2 Latency Effects 4-13
4.4.2.1 Latency 1 - DA Instruction - Memory Writes 4-14
4.4.2.2 Latency 2 - Accumulator as Multiplier Input 4-14
4.4.2.3 Latency 3 - Branching 4-14
4.4.2.4 Latency 4 - Conditional Branching on DA Conditions 4-15
4.4.3 Common Programming Techniques 4-15
4.4.3.1 Subroutine Nesting 4-15
4.4.3.2 Passing Parameters 4-16
4.4.3.3 Quick Interrupt Facility 4-17
4.5 Common Signal Processing Functions 4-17
4.5.1 Standard Signal Processing Functions 4-17
4.5.2 Nonlinear Functions 4-18
4.6 Detailed Description of Instruction Set 4-20

4. INSTRUCTION SET

The DSP3210 has two general types of instructions that correspond to the two execution units: data arithmetic (DA) instructions and control arithmetic (CA) instructions. Primarily, DA instructions perform 32-bit floating-point multiply/accumulate operations for signal processing algorithms. Other DA instructions convert the DSP's internal floating-point data to and from each of the following types: 8-, 16-, or 32-bit integer, μ-law, A-law, or single-precision IEEE floating-point format. The CA instructions perform microprocessor operations such as 16- and 32-bit integer arithmetic and logic functions, conditional branching, and moving data. Instruction bit encodings are contained in Section 10 - Instruction and Register Set Encodings.

Figure 4-1 shows the register programming model of the DSP3210. It consists of twenty-two 32-bit general purpose registers, one register hardwired to zero, a 32-bit program counter, four 40-bit floating-point accumulator registers, and five IO control registers. These registers are summarized below:

- General-purpose CAU 32-bit registers, r1—r22
- CAU register hard-wired to zero, r0
- 32-bit program counter, pc, and associated interrupt shadow register, pcsh
- 40-bit floating-point accumulator registers, a0—a3, and associated interrupt shadow registers
- Control registers
  - –Processor status register, ps, and associated shadow register
  - –Exception mask register, emr
  - –Processor control word, pcw
  - –DAU control register, dauc, and associated shadow register
  - –Clip-test register, ctr, and associated shadow register

Although r1—r22 can be used for general purposes, under certain circumstances these registers are designated to be used in specific ways:

Register	Name	Meaning
r1—r14	rP	DA instruction memory reference (X, Y, Z fields) pointer registers.
r15—r19	rI	DA instruction memory reference (X, Y, Z fields) increment registers.
r20	r20	Used by error exception facility to store the old pc.
r21	sp	Stack pointer.
r22	evtp	Used as the pointer to the exception vector table.

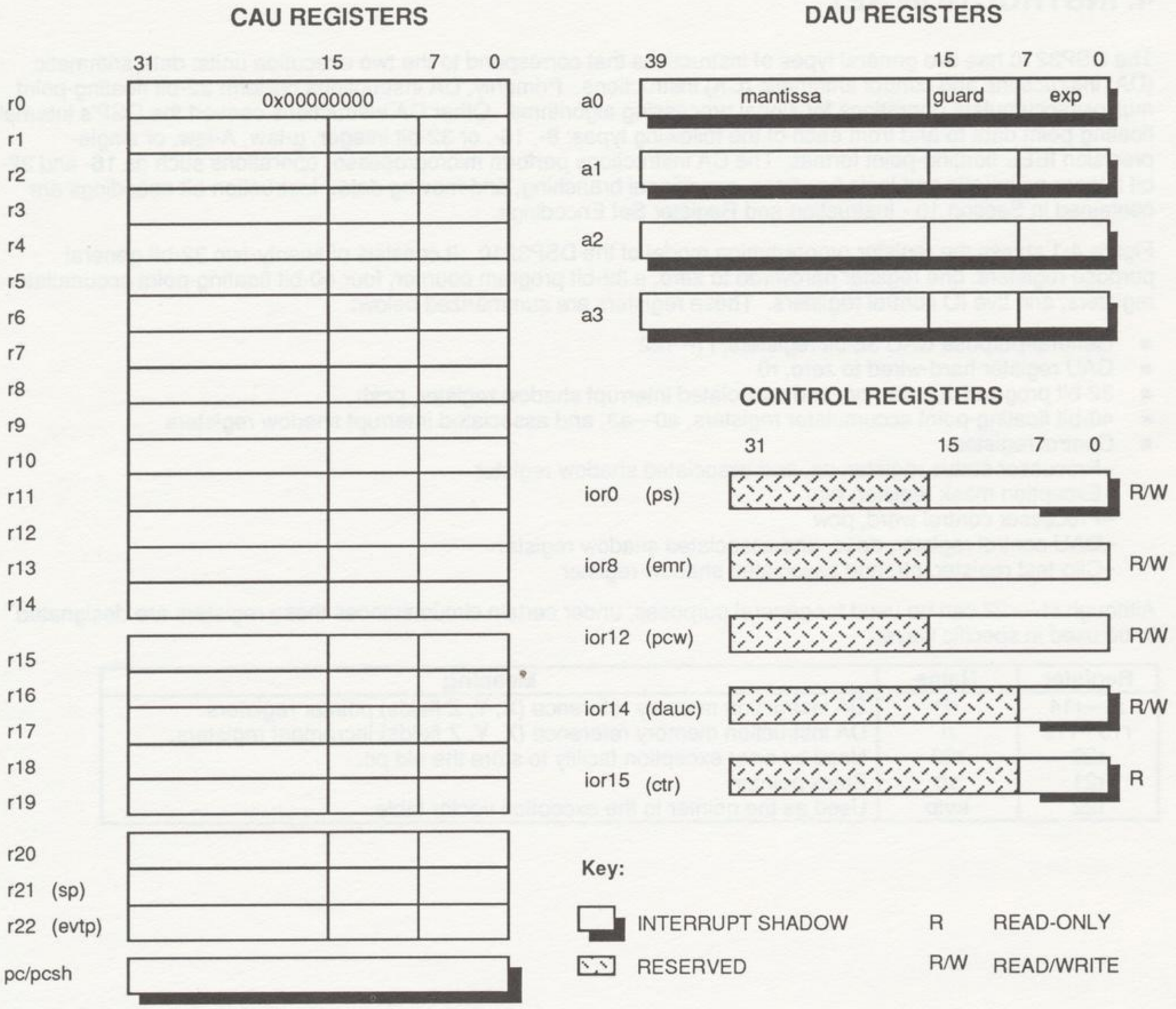


Figure 4-1 Register Programming Model

4.1 Flags

The DSP3210 has internal flags that are affected by the results of certain DA, or CA instructions and certain I/O events. These flags may be tested by conditional instructions and may also be read from the ps register. Table 4-1 lists the flags and their meaning.

DSP3210 instructions and the flags affected by each instruction are specified in sections 4.2 DA Instructions and 4.3 CA Instructions. A zero (0) shown in place of a flag means that the flag is always made zero; a dash (-) in place of a flag means that the flag is unaffected by the instruction.

Table 4-1 DSP3210 Flags

DAU FLAGS				
Flag	**True**	**False**	**Meaning (Flag = 1)**	**Definition**
N	alt	age	Result is negative	sign bit = one
Z	aeq	ane	Result is zero	mantissa = zero
V	avs	avc	Result overflowed	|DAU result | > 3.40282E38
U	aus	auc	Result underflowed	|DAU result | < 5.87747E-39

CAU FLAGS				
Flag	**True**	**False**	**Meaning (Flag = 1)**	**Definition**
n	mi	pl	Result is negative	(16-bit) n=b31 (bit 31 of ALU result) (16-bit) n=b15 (bit 15 of Barrel Shifter result) (32-bit) n=b31
z	eq	ne	Result is zero	(16-bit) z=b31 + b30 + ... b1 + b0 (+=OR) (32-bit) z=b31 + b30 + ... b1 + b0
c	cs	cc	Carry or borrow out of MSB	(16-bit) c=b15c (carry out of ALU bit 15) (32-bit) c=b31c
v	vs	vc	Result overflowed	(16-bit) v=b14c^b15c (^=exclusive OR) (32-bit) v=b30c^b31c

I/O FLAGS				
Flag	**True**	**False**	**Meaning (Flag = 1)**	**Definition**
i	ibf	ibe	Serial input buffer full	IBF = 1
o	obf	obe	Serial output buffer full	OBE = 0
s	sys	syc	SY (I/O sync) set	SY = 1
b	fbs	fbc	SIO frame boundary	First OBE after SY high-to-low transition
r	ir0s	ir0c	Interrupt 0 set	IR0N = 1
R	ir1s	ir1c	Interrupt 1 set	IR1N = 1

4.2 DA Instructions

The data arithmetic (DA) instructions are divided into two functional groups: multiply/accumulate and special funtions.

4.2.1 DA Instructions - Multiply/Accumulate

DA multiply/accumulate instructions perform 32-bit floating-point multiply/accumulate operations for signal processing algorithms. The DA multiply/accumulate instructions are listed in Table 4-2. Replacements for the operands are listed in Table 4-3.

Note: { } and [ ] are not part of the instruction syntax. Parentheses ( ) are part of the syntax and must appear where shown in an instruction. Lower-case letters are part of the syntax and upper-case letters are replaced by immediate data or by a register name (see tables following each instruction group). A detailed description of each instruction is available in Section 4.6 under the Instruction Reference name.

Table 4-2 DA Multiply/Accumulate Instructions

Instruction	DAU Flags	Instr. Reference	Description
[Z=] aN = [–]aM {+, –}Y*X	NZVU	FMULT-ACC-STORE	The product of the X and Y fields is added/subtracted to/from the accumulator aM and the result is stored in accumulator aN. The result can also be output according to the Z field.
aN = [–]aM {+, –}(Z=Y)*X	NZVU	FMULT-ACC-TAP	The Y field operand is output according to the Z field. The product of the X and Y fields is added to accumulator aM and the sum is stored in accumulator aN.
[Z=] aN = [–]Y {+, –}aM*X	NZVU	FMULT-ADD-STORE	The product of the X field and accumulator aM is added/subtracted to/from the Y field. The result is placed in accumulator aN and can also be output according to the Z field.
[Z=] aN = [–]Y *X	NZVU	FMULT-STORE	The product of the X and Y fields is added/subtracted to/from zero. The result is stored in accumulator aN and can also be output according to the Z field.
aN = [–] (Z=Y)*X	NZVU	FMULT-TAP	The value of the Y field is output according to the Z field. The product of the X and Y fields is stored in accumulator aN.
[Z=] aN = [–]Y {+, –} X	NZVU	FMULT-STORE	The sum or difference of the Y and X fields is stored in accumulator aN and the result can also be output according to the Z field.
[Z=] aN = [–]Y	NZVU	FLOAD-STORE	The value of the Y field is placed in accumulator aN and can also be output according to the Z field.
aN = [–] (Z=Y) {+, –} X	NZVU	FADD-TAP	The sum or difference of the X and Y fields is stored in accumulator aN, and Y is output according to the Z field.

Table 4-3 Replacement for DA Multiply/Accumulate Instructions

Replace	Value*	Meaning
aN, aM	a0-a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI,	32-bit memory location, or
	a0-a3	one of the four DAU accumulators
Z	*rP, *rP++, *rP--, *rP++rI	32-bit memory location

* rP refers to r1—r14 and is used as a memory pointer. rI refers to r15—r19 and is used as an increment register.

4.2.2 DA Instructions - Special Function

DA special function instructions perform data type format conversions, rounding, conditional accumulator loads, and compute a reciprocal seed. The DA special function instructions are listed in Table 4–4. Replacements for the operands are listed in Table 4-5.

Table 4–4 DA Special Function Instructions

Instruction	DAU Flags	Instr. Reference	Description
[Z=] aN = ic(Y)	NZ00	IC	Input conversion μ-law, A-law, 8-bit linear to float.
[Z=] aN = oc(Y)	—	OC	Output conversion float to μ-law, A-law, 8-bit linear.
[Z=] aN = float16(Y)	NZ00	FLOAT16	16-bit integer to float.
[Z=] aN = float32(Y)	NZ00	FLOAT32	32-bit integer to float.
[Z=] aN = int16(Y)	—	INT16	Float to 16-bit integer (round or truncate, dauc[4]).
[Z=] aN = int32(Y)	—	INT32	Float to 32-bit integer (round or truncate, dauc[4]).
[Z=] aN = round(Y)	NZVU	ROUND	Round to nearest, float(40) to float(32).
[Z=] aN = ifalt(Y)	—	IFALT	Conditional assignment/memory write.
[Z=] aN = ifaeq(Y)	—	IFAEQ	Conditional assignment/memory write.
[Z=] aN = ifagt(Y)	—	IFAGT	Conditional assignment/memory write.
[Z=] aN = dsp(Y)	NZVU	DSP	IEEE to DSP format conversion.
[Z=] aN = ieee(Y)	—	IEEE	DSP to IEEE format conversion.
[Z=] aN = seed(Y)	NZ0U	SEED	32-bit to 32-bit reciprocal seed.

Table 4-5 Replacement for DA Special Function Instructions

Replace	Value*	Meaning
aN	a0-a3	One of the four DAU accumulators
Y†	*rP, *rP++, *rP--, *rP++rI,	8-, 16-, or 32-bit memory location, or
	a0-a3	one of the four DAU accumulators
Z	*rP, *rP++, *rP--, *rP++rI	8-, 16-, or 32-bit memory location

* rP refers to r1—r14 and is used as a memory pointer. rI refers to r15—r19 and is used as an increment register.
† Y may not be a0—a3 for the *dsp* special function.

4.3 CA Instructions

The control arithmetic (CA) instructions perform 16- and 32-bit integer operations. These operations include generating addresses, performing integer data and logical functions, instruction flow control, and data movement. The instructions are grouped into three major headings: control, arithmetic and logic, and data move.

4.3.1 CA Instructions - Control

The CA control instructions alter the ordinary flow of control from one instruction to the next. They include branches, loop counter test, subroutine call and return, return from interrupt, do loops, and wait for interrupt. Instructions can branch based on CA, DA, or I/O conditions. The condition for the branch is determined by the state of the flags resulting from the most recently calculated CA or DA instruction or I/O event. The CA control instructions have no effect on the flags. The CA control instructions are listed in Table 4-6. Replacements for the COND field are listed in Table 4-7.

Table 4-6 CA Control Instructions

Instruction	Instr. Reference	Description
if (COND) goto {N, rB, rB+N}*	GOTO-COND	Conditional branch based on flags
if (rM-->=0) goto {N, rB, rB+N}*	GOTO-LOOP	Conditional branch using loop counter
goto {N, rB, rB+N, M, rB+M}*	GOTO	Unconditional branch
nop	NOP	No operation
call {N, rB, rB+N, M} (rM)*	CALL	Call subroutine
return (rM)	RETURN	Return from subroutine
do K, {L, rM}	DO	Do next K+1 instruction(s) L+1 (or rM+1) time(s). K=0, 1, 2, ...127; L=rM=0, 1, 2, ...2047
dolock K, {L, rM}	DO	*dolock* signals interlocked bus accesses.
doblock {L, rM}	DO	*doblock* signals quad-word transfers.
ireturn	IRETURN	Return from interrupt
sftrst	SFTRST	Soft Reset; Changes error level to base level; encoded as spc=(byte)r0
waiti	WAITI	Wait for interrupt; encoded as spc=(long)r0

* When using pc as rB, the conditional or unconditional goto and call instructions may be written with the syntax *pcgoto label* or *pccall label*, respectively. The linker will compute the appropriate N (offset) value to encode.

where:
rB = pc, r0-r22
rM = r1-r22
N = 16-bit signed integer
M= 24-bit unsigned integer
COND = one of the DSP3210 Condition Codes (See Tables below)

Table 4-7 Replacement for COND

(COND) - CAU Conditions		
Condition	**Flag***	**Meaning**
false	Not applicable	Condition is always false.
true	Not applicable	Condition is always true.
pl	n=0	Result is nonnegative (plus)
mi	n=1	Result is negative (minus)
ne	z=0	Result not equal to zero
eq	z=1	Result equal to zero
vc	v=0	Overflow clear, no overflow
vs	v=1	Overflow set, overflowed
cc	c=0	Carry clear, no carry
cs	c=1	Carry set, carry
ge	n^v=0	Greater than or equal to zero
lt	n^v=1	Less than zero
gt	z | (n^v)=0	Greater than zero
le	z | (n^v)=1	Less than or equal to zero
hi	c | z=0	Greater than (unsigned number)
ls	c | z=1	Less than (unsigned number)
(COND) - DAU Conditions		
Condition	**Flag**	**Meaning**
ane	Z=0	Not equal to zero
aeq	Z=1	Equal to zero
age	N=0	Greater than or equal to zero
alt	N=1	Less than zero
avc	V=0	Overflow clear, no overflow
avs	V=1	Overflow set, overflowed
auc	U=0	Underflow clear, no underflow
aus	U=1	Underflow set, underflowed
agt	N | Z=0	Greater than zero
ale	N | Z=1	Less than or equal to zero
(COND) - IO Conditions		
Condition	**Flag**	**Meaning**
ibe	ibf=0	Input buffer empty
ibf	ibf=1	Input buffer full
obe	obe=1	Output buffer empty
obf	obe=0	Output buffer full
syc	sy=0	Sync signal low
sys	sy=1	Sync signal high
fbc	fb=0	Serial frame boundary clear
fbs	fb=1	Serial frame boundary set
ir0s	ir0n=1	IR0N pin is logic one (1).
ir0c	ir0n=0	IR0N pin is logic zero (0).
ir1s	ir1n=1	IR1N pin is logic one (1).
ir1c	ir1n=0	IR1N pin is logic zero (0).

* Symbol interpretation: ^ = XOR; |= OR

4.3.2 CA Instructions - Arithmetic/Logic

The CA arithmetic/logic instructions perform integer arithmetic and logical operations in the CAU. Instructions of this type do not have memory references. The operands for these instructions are contained in the CAU registers or supplied in the instruction as immediate data. Operands that are in memory must first be moved to the CAU register array using a data move instruction (see 4.3.3 CA Instructions - Data Move). The size of the operation can be specified to be long (32-bits), or short (16-bits). **Long is the default and does not have to be specified.**

Table 4-8 lists the CA Arithmetic/Logic instructions.

Symbols in Table 4-8 can be replaced with:

rD = r0—r22
rD1 = r0—r20, r22
rS, rS1, rS2 = r0—r22
rS3 = r0—r22, pc, pcsh
N = 16-bit signed integer
COND = one of the DSP3210 Condition Codes (see Table 4-7 Replacement for COND)

Table 4-8 CA Arithmetic/Logic Instructions

Instruction	CAU Flag	Instr. Reference	Description
rD = {(long), (short)} rS3+N	nzvc	ADD	Three operand add with 16-bit sign-extended immediate
rD = (long) rS <<| N	nz00	SHIFT-OR	Shift 16-bit immediate left 16 places and OR
rD = {(long), (short)} rD - N	nzvc	SUBTRACT	Right subtract with immediate
rD = {(long), (short)} N - rD	nzvc	SUBTRACT	Left subtract with immediate
rD = {(long), (short)} rD & N	nz00	ADD	Logical AND with immediate
rD = {(long), (short)} rD | N	nz00	OR	Logical OR with immediate
rD = {(long), (short)} rD ^ N	nz00	EXCLUSIVE-OR	Logical XOR with immediate
rD = {(long), (short)} rD # N	nz0c	ADD-CARRY REV	Carry reverse add with immediate
rD = {(long), (short)} rD << N	nz00	SHIFT LEFT	Left shift by N (0≤N≤31)
rD = {(long), (short)} rD >> N	nz00	SHIFT RIGHT	Logical right shift by N (0≤N≤31)
rD = {(long), (short)} rD $>> N	nz00	SHIFT RIGHT-ARITH	Arithmetic right shift by N (0≤N≤31)
{(long), (short)} rD - N	nzvc	COMPARE	Compare with immediate
{(long), (short)} rD & N	nz00	BIT TEST	Bit Test with immediate
[if(COND)] rD = {(long), (short)} rS1 + rS2	nzvc	ADD	Conditional 3-register add
[if(COND)] rD = {(long), (short)} rS1 - rS2	nzvc	SUBTRACT	Conditional 3-register subtract
[if(COND)] rD = {(long), (short)} rS1 & rS2	nz00	ADD	Conditional 3-register AND
[if(COND)] rD = {(long), (short)} rS1 &~ rS2	nz00	AND-COMPLEMENT	Conditional 3-register AND with complement
[if(COND)] rD = {(long), (short)} rS1 | rS2	nz00	OR	Conditional 3-register OR
[if(COND)] rD = {(long), (short)} rS1 ^ rS2	nz00	EXCLUSIVE OR	Conditional 3-register XOR
[if(COND)] rD = {(long), (short)} rS1 # rS2	nz0c	ADD-CARRY REV	Conditional 3-register carry reverse add
[if(COND)] rD = {(long), (short)} rS1 << rS2	nz00	SHIFT LEFT	Conditional left shift by rS2 (0≤rS2≤31)
[if(COND)] rD = {(long), (short)} rS1 >> rS2	nz00	SHIFT RIGHT	Conditional logical right shift by rS2 (0≤rS2≤31)
[if(COND)] rD = {(long), (short)} rS1 $>> rS2	nz00	SHIFT RIGHT-ARITH	Conditional arithmetic right shift by rS2 (0≤rS2≤31)
[if(COND)] {(long), (short)} rS1 - rS2	nzvc	COMPARE	Conditional compare
[if(COND)] {(long), (short)} rS1 & rS2	nz00	BIT TEST	Conditional bit test
[if(COND)] rD = {(long), (short)} rS/2	nz00	SHIFT RIGHT-ARITH	Conditional arithmetic right shift by 1
[if(COND)] rD = {(long), (short)} rS>>1	nz00	SHIFT RIGHT	Conditional logical right shift by 1
[if(COND)] rD = {(long), (short)} rS<<1	nz00	SHIFT LEFT	Conditional left shift by 1
[if(COND)] rD = {(long), (short)} rS*2	nzvc	ADD	Conditional multiply by 2
[if(COND)] rD = {(long), (short)} rS>>>1	nz0c	ROTATE RIGHT	Conditional rotate right through carry
[if(COND)] rD = {(long), (short)} rS<<<1	nzvc	ROTATE LEFT	Conditional rotate left through carry
[if(COND)] rD = {(long), (short)} -rS	nzvc	SUBTRACT	Conditional negate
[if(COND)] rD = {(long), (short)} rS	nzvc	ADD	Conditional assignment
[if(COND)] rD1 = {(long), (short)} rS {+, –}1	nzvc	INCR/DECR	Conditional increment/ decrement
[if(COND)] sp = (long) sp {++, –-}	nzvc	INCR/DECR	Conditional increment/ decrement of stack pointer by 4 (Interrupts are disabled for one instruction cycle)

4.3.3 CA Instructions - Data Move

The CA data move instructions are divided into four types: (1) immediate data sets of CAU registers, (2) data transfers between memory and CAU registers, (3) data transfers between IO registers and CAU registers, and (4) data transfers between IO registers and memory. The data can be moved as 8-bit (byte), 16-bit (half-word), or 32-bit (word). Table 4-9 lists the CA Data Move instructions and Table 4-10 shows the operand replacements.

When loading CAU registers from memory or an IO register, the size of the transfer may be specified to be 8-bit (byte, hbyte, char), 16-bit (short, ushort), or 32-bit (long). **Long is the default and does not have to be specified.** The affect on CAU register bits for each operation is described below:

Destination CAU Register Bits Affected					
Type Specified	**31—24**	**23—16**	**15—8**	**7—0**	**Description**
(byte)	0	0	0	DATA	Zero-extend to upper 24-bits
(hbyte)	0	0	DATA	0	Zero-fill bits 7-0, zero-extend upper 16-bits
(char)	S	S	S	DATA	Sign-extend bit 7 to upper 24-bits
(short)	S	S	DATA	DATA	Sign-extend bit 15 to upper 16-bits
(ushort)	0	0	DATA	DATA	Zero-extend to upper 16-bits
(long)	DATA	DATA	DATA	DATA	Load bits 31—0.

When storing CAU registers in memory or IO registers, the size of the operation may be specified to be 8-bit (byte, hbyte), 16-bit (short), or 32-bit (long). **Long is the default and does not have to be specified.** When the destination is an IO register, only the number of IO register bits specified by the size of the transfer are affected (8, 16, 32). The size specified for the transfer must be the same as the size of the register implemented in the DSP3210. The difference between (byte) and (hbyte) is the CAU register bits selected for the operation. The CAU register bits selected for each operation are described below:

Source CAU Register Bits Selected					
Type Specified	**31—24**	**23—16**	**15—8**	**7—0**	**Description**
(byte)	X	X	X	DATA	Store bits 7—0
(hbyte)	X	X	DATA	X	Store bits 15—8
(short)	X	X	DATA	DATA	Store bits 15—0
(long)	DATA	DATA	DATA	DATA	Store bits 31—0

When loading IO registers from memory, the size of the transfer may be specified to be 8-bit (byte), 16-bit (short), or 32-bit (long). Only the number of IO register bits specified by the size of the transfer are affected (8, 16, 32). When storing IO register to memory, the size of the transfer may be specified to be 8-bit (byte), 16-bit (short), or 32-bit (long). The size specified for the transfer must be the same as the size of the register implemented in the DSP3210.

Table 4-9 CA Data Move Group Instructions

Instruction	CAU Flag	Instr. Reference	Description
rD= (short) N	nz00	SET	Reg Set- 16-bit sign extended to 32-bits
rD= (ushort24) M	—	SET24	Reg Set- 24-bit zero extended to 32-bits
{MEM, *L} ={(byte), (hbyte), (short), (long)} rS	—	STORE	Reg Store- MEM, *L are 8-, 16-, or 32-bit
rD = {(char), (byte), (hbyte)} {MEM, *L}	nz00	LOAD	Reg Load- MEM, *L are 8-bits
rD = {(short), (ushort)} {MEM, *L}	nz00	LOAD	Reg Load- MEM, *L are 16-bits
rD = (long) {MEM, *L}	nz00	LOAD	Reg Load- MEM, *L are 32-bits
iorD = {(byte), (hbyte), (short), [(long)]} rS	—	STORE	Reg Store- iorD are 8--, 16-, or 32-bit
rD = {(char), (byte), (hbyte)} iorS	nz00	LOAD	Reg Load- iorS are 8-bits
rD = {(short), (ushort)} iorS	nz00	LOAD	Reg Load- iorS are 16-bits
rD = (long) iorS	nz00	LOAD	Reg Load- iorS are 32-bits
MEM = {(byte), (short), (long)} iorS	—	STORE-IOR	IO Reg to Memory move
iorD = {(byte), (short), (long)} MEM	—	LOAD-IOR	Memory to IO Reg move

Table 4-10 Replacement for CA Data Move Instructions

	Value	Meaning
rS	r0-r22, pc, pcsh	One of 22 CAU registers, pc, or pc shadow register
rD	r0-r22	One of 22 CAU registers
MEM	*rP, *rP++, *rP--, *rP++rI	8-, 16-, or 32-bit memory location rP is a memory pointer (rP=r0—r22); when using the sp (r21), the size of the transfer must be long (++ and -- are +4 and –4), rI is an increment register (rI=r0-r22)
iorS	ior#	One of the IO registers. The size of the selected register and transfer size must be the same. (#=0, 8, 10, 12, 14, 15)
iorD	ior#	One of the IO registers. (#=0, 8, 10, 12, 14, 15)
N	16-bit number	2's complement number
L	16-bit number	Unsigned number–used as a direct address
M	24-bit number	Unsigned number

4.4 DSP3210 Programming

This chapter explains how to combine the instructions presented in Chapter 3 to form valid DSP3210 programs. The programmer must be aware of the latencies that occur because of the pipelining of the device and the restrictions on the order of certain instructions. This chapter presents this information, along with a set of programming examples for encoding common functions.

4.4.1 Restrictions

This section describes three sequences of instructions that are not allowed. The assembler recognizes these sequences and generates a fatal error.

4.4.1.1 Restriction 1 - Z-Field Pointer

The instruction immediately following a DA instruction with a Z-field write, in which the Z-field pointer is post-modified and cannot reference the Z-field pointer except as another Z-field pointer of a DA instruction. This is a fatal error, and the assembler produces an error message. In the event that an instruction with a post-modified Z-field pointer follows a branch instruction, the assembler produces a warning message.

Example 1:

```
*r1++ = a2 = a2 + *r3 * a1
 r1 = r1 + 12
```

ERROR: Using reg from preceding inst's Z-field pointer

Example 2:

```
*r1-- = a2 = a2 + *r3 * a1
a2 = a2 + *r1 * a1
```

ERROR: Using reg from preceding inst's Z-field pointer

Example 3:

```
 if (le) goto r20
*r1 = a2 = a2 + a3 * a1
```

Warning: Use output pointer reg only as output pointer in DAU inst atdest after branch instruction.

4.4.1.2 Restriction 2 - CAU and IO Register Store

A CAU or IO register store instruction cannot immediately follow a DA instruction (multiply/accumulate or special function) that references memory in the Y field. This is a fatal error, and the assembler produces an error message. In the event that a DA instruction with a memory reference in the Y field immediately follows a branch instruction, the assembler produces a warning message. The affected CAU and IO Register Store instructions are:

```
{MEM, *L} ={(byte), (hbyte), (short), (long)} rS
iorD = {(byte), (hbyte), (short), [(long)]} rS
MEM = {(byte), (short), (long)}  iorS
```

Recall that one form of the DA instructions is:

```
[Z] = aN = [–]aM {+, –} Y ** X
```

Example 1:

```
a2 = a2 + *r2 *a3
*r4 = r5
```

ERROR: CAU reg or IO reg store can't follow DAU inst with 'Y' mem ref

Example 2:

```
a0 = ic(ibuf)
*r4 = r5
```

ERROR: CAU reg or IO reg store can't follow DA inst with 'Y' IO register ref

Example 3:

```
if (ge) goto  r3
a2 = a2 + *r2 * a3
```

Warning: branch dest can't start with CAU reg store inst.

4.4.1.3 Restriction 3 - CAU and IO Register Load

A CAU register (r0—r22) that is being loaded from memory or from an IO register, or an IO register (ps, emr, pcw, dauc) that is being loaded from memory cannot be referenced in the following instruction. The only exception is that a CAU register can be referenced as the Z-field pointer of a DA instruction. Note that the flags that result from this load cannot be referenced in the following instruction either. This is a fatal error, and the assembler produces an error message. If a CAU or IO register load follows a branch instruction, the assembler produces a warning message. A dauc load from memory must be followed by a CA instruction. The affected CAU and IO Register Load instructions are:

```
rD = {(char), (byte), (hbyte), (short), (ushort), (long)} {MEM, *L}
iorD = {(byte), (short), (long)} MEM
```

Example 1:

```
r3 = *r2
r4 = r3 + r7
```

ERROR:Cannot reference reg loaded in previous inst except as Z-field ptr

Example 2:

```
if (ne) goto r3 + 8
r4 = *r7++
```

Warning: branch destination can't start with inst referencing reg being loaded.

4.4.2 Latency Effects

The pipelined execution of Data Arithmetic DSP3210 instructions results in four latency effects that are listed in this section. See 8.2 Data Arithmetic Unit (DAU) for an explanation of the DAU pipeline.

4.4.2.1 Latency 1 - DA Instruction - Memory Writes

When a DA instruction specifies a write to memory, the value written is not available to be read from that location until four instructions later (three-instruction latency).

Example:

```
I1   *r3 = a0 = a0
I2   *r3 = a3 = a3
I3   .
I4   .
I5      a1 = *r3
```

The value read from memory in instruction five (I5) is the value written in I1, not I2.

4.4.2.2 Latency 2 - Accumulator as Multiplier Input

When an accumulator, a0-a3, is used as an input to the multiplier, its value is established no sooner than three instructions prior to the multiply instruction (2-instruction latency). Note that this also applies to an accumulator using the X field of an instruction of the form:

[Z =] aN = [–]Y {+, –} X

Example:

```
I1   a0 = a0 + *r1 * *r2
I2   a0 = a0 + a1
I3   .
I4   a2 = a0 * a0
I5   a1 = a2 + a0
```

The value of a0 used in I4 is calculated in I1. The value of a0 used in I5 is calculated in I2. Note that the value of a2 used in I5 is calculated in I4 since there is no latency effect on accumulators used as inputs to the adder.

4.4.2.3 Latency 3 - Branching

When a CA Control Group instruction of the form *if()goto, call, return, goto* is executed, the instruction immediately following is also executed before the branch occurs. This is commonly referred to as a delayed branch. The *ireturn* instruction is different, and execution of the base-level program resumes in the following instruction cycle.

Example 1:

```
I1   if (eq) goto over
I2   r1 = 3
I3   .
```

The instruction I2 is executed even if the condition is true and the branch is taken. If this is undesirable, a *nop* can be placed after the branch instruction, or if possible, the instructions can be rearranged. Because of this latency, a complex situation arises if successive branch instructions are coded as in the following example.

Example 2:

```
I1   goto A
I2   goto B
       .
       .
A:     .
       .
       .
B:     .
       .
C:     .
       .
```

The order of execution is I1, I2, A, B. If the instruction at A is not a goto, execution continues from B. If the instruction at A is goto C, the order of execution is I1, I2, A, B, C, and execution continues from C. Successive branch instructions are useful in some applications.

4.4.2.4 Latency 4 - Conditional Branching on DAU Conditions

A DAU condition tested by a conditional branch or conditional arithmetic/logic instruction is established by the last DA instruction that affects DAU flags no sooner than four instructions prior to the test (3-instruction latency).

Example:

```
I1   a0 = a0 + a1
I2   a2 = a0 * a2
I3   .
I4   .
I5   if (agt) goto next
I6   .                  /* latent instruction */
```

The condition tested in I5 is established by the instruction at I1, not I2.

Because of this latency effect, use the zero-latency *ifalt, ifagt, and ifaeq* functions where possible (see 4.2.2 DA Instructions - Special Functions). The DA condition tested by these conditional accumulator loads is established by the last DA instruction that affected the DAU flags.

4.4.3 Common Programming Techniques

The following are programming techniques common to the DSP3210.

4.4.3.1 Subroutine Nesting

The following example is intended only as an illustration of DSP3210 programming technique. The convention adopted is that the stack frame grows from high addresses toward low addresses, and that the stack pointer contains the first empty location. The latter uncommon convention saves one instruction of overhead in the push operation because the DSP3210 has only postincrement and not preincrement addressing.

The following example shows how to nest subroutines:

```
        call subr1 (r1)          /* ret1 is stored in r1 */
        .                        /* latent instruction   */
ret1:   .
        .
        .
subr1:  .
        .
        .
        *sp-- = r1               /* push ret1 onto stack using stack pointer*/
        call subr2 (r1)          /* ret2 is stored in r1 */
        .                        /* latent */
ret2:   .
        .
        .
        return (r1)              /* go to ret1 */
        .                        /* latent     */
subr2:  .
        .
        .
        sp = sp++                /* sp now points to top of stack     */
        r1 = *sp                 /* restore linkage pointer (ret1) */
        return (r1)              /* go to ret2  */
        .                        /* latent */
```

Alternatively, if only a small level of nesting is required and sufficient CAU registers are available, the different return addresses can simply be saved in different CAU registers.

4.4.3.2 Passing Parameters

Parameters may be passed to a subroutine if they immediately follow the latent instruction associated with the call:

```
1.  call subr (r1);
2.  .                        /* latent instruction */
3.  parameter #1 for subr
4.  parameter #2 for subr
5.  parameter #3 for subr
6.  parameter #4 for subr
7.  .
etc.
```

The subroutine should use *r1++ to retrieve the parameters. After all four parameters are retrieved, a *return (r1)* is executed that resumes execution at line 7.

4.4.3.3 Quick Interrupt Facility

Each entry in the interrupt vector table may contain two instructions. In general, these may be used for a *goto* and the delayed branch latent instruction. Alternatively, by using these two locations for an instruction followed by an *ireturn*, an interrupt routine with an overhead of three instruction cycles can be accomplished.

The interrupt operation performed by the DSP3210 while processing the instruction stream I1, I2, and I3 is as follows:

```
I1                      /* program instruction #1                          */
I2                      /* program instruction #2                          */
goto evtp + offset      /* overhead branching to interrupt                 */
nop                     /* vector table plus the offset is computed by the DSP3210 based on  */
                        /* the interrupt source                            */
IR1                     /* useful interrupt routine instruction            */
ireturn                 /* no latent instruction executed after ireturn    */
I3                      /* program instruction #3                          */
```

IR1 is the useful interrupt routine instruction. **IR1 cannot be a DA instruction.**

4.5 Common Signal Processing Functions

This section presents instruction sequences for common functions performed in digital signal processing. Many more example programs are contained in the DSP3210 Application Software Library Reference Manual. **All instructions require one 32-bit word for storage and execute in one instruction cycle.** Note that the time to execute one instruction is affected by conflict and external waitstates.

4.5.1 Standard Signal Processing Functions

The following functions illustrate the source code syntax and programming techniques for the DSP3210. All instruction sequences may be preceded by the *do* instruction for repeated execution.

(a) Convolution (FIR filter)

$$y(n) = \sum_{k=0}^{k=N-1} h[k] * x(n-k)$$

```
(1)     a1 = a1 + (*r3++ = *r4++) * *r2++
```

(b) LMS update for adaptive filtering

$$h(k+1) = h(k) + 2 * \mu * e(k) * x(k)$$

```
(1)     *r2++ = a1 = *r2 + a0 * *r4++
```

(c) Second-order biquad IIR filter

$$w_i(n) = v_{i-1}(n) - d(i,1)s_{i1}(n-1) - d(1,2)s_{i2}(n-1)$$
$$v_i(n) = n(i,0)w_i(n) + n(i,1)s_{i1}(n-1) + n(i,2)s_{i2}(n-1)$$
$$s_{i1}(n) = w_i(n)$$
$$s_{i2}(n) = s_{i1}(n-1)$$

```
(1)     a0 = * r5 * *r2++                       /* input =* K1     */
(2)     a0 = a0 - *r3++ * *r2++                 /* d(i, 1) term    */
(3)     *r4++ = a0 = a0 - *r3-- * *r2++         /* d(i, 2) term    */
(4)     a0 = a0 + (*r4++ = *r3++) * *r2++       /* n(i, 1) term    */
(5)     *r6 = a0 = a0 + *r3++ * *r2++           /* n(i, 2) term    */
```

(d) FFT Butterfly

```
(1)     a0 = *r1++r17 + a2 * *r3++r17          /* T = R(i) * Ur * R(k)      */
(2)     *r11++r17 = a0 = a0 - *r3++r19 * a3    /* R(i) = T = T - Ui * I(k)  */
(3)     a1 = *r1++r19 + a3 * *r3++r17          /* T = I(i) + Ui * R(k)      */
(4)     *r12++r17 = a0 = -a0 + *r1++r17 * *r6  /* (Rk) = -T + 2 * R(i)      */
(5)     *r11++r15 = a1 = a1 + *r3++r17 * *a2   /* I(i) = T = T + Ur * I(k)  */
(6)     *r12++r15 = a1 = -a1 + *r1++r15 * *r6  /* I(k) + -T + 2 * I(i)      */
```

(e) In-place bit reversal

```
A:      r1 - r2                                /* compare true and bit-reversed  */
        if(ge) pcgoto B                        /* exchange if r1 < r2            */
        r5 = r1                                /* latent instruction             */
        *r1++r17 = a0 = *r2++r17               /* complex exchange               */
        *r1++r19 = a0 = *r2++r19
        *r2++r19 = a0 = *r5++r19
        *r2++r19 = a0 = *r5++r19
B:      r1 = r1 + 4
        if(r18-- >= 0) pcgoto A
        r2 = r2 # r16                          /* # is carry reverse add         */
```

(f) Inverse using the seed instruction

```
(1)     a3 = seed(*r1)                         /* r1 points to input table       */
(2)     r3 = _K                                /* r3 points to constants         */
(3)     r4 = result                            /* r4 points to output table      */
(4)     a0 = *r3++ + a3 * *r1                  /* a0 = -1.6 - x * y              */
(5)     a1 = *r3++ * a3                        /* a1 = 0.8 * x                   */
(6)     nop
(7)     a0 = *r3++ + a0 * a0                   /* a0 = 31 - 3.2 * x * y + x^2 * y^2 */
(8)     a2 = a1 * *r1++                        /* a2 = 0.8 * x * y               */
(9)     nop
(10)    a0 = a0 * a1                           /* a0 = 2.8 - 2.6 * x^2 * y + 0.8 *  */
                                               /* x^3 * y^2                      */
(11)    a1 = *r3--r17 + a0 * a2                /* a1 = 2.8 * x * y - 2.6 * x^2 *    */
                                               /* y^2 + 0.8 * x^3 * y^3          */
(12)    nop
(13)    nop
(14)    *r4++ = a0 = a0 - a1 * a0              /* result is in a0                */

_K: float -1.60138661
    float 0.82371354
    float 0.85221935
    float -1.0
```

4.5.2 Nonlinear Functions

Several nonlinear functions are programmed using the *ifalt or ifagt* special functions (see 4.2.2 DA Instructions Special Functions). The DSP3210 programs for some of these nonlinear functions are included here.

(a) Absolute Value or Full-Wave Rectification

```
(1)     a0 = -a2
(2)     a2 = ifagt(a0)
```

The value in a2 is replaced by its absolute value.

(b) Signum

```
(1)     a0 = *r1
(2)     a1 = *r3
(3)     *r3 = a0 = ifalt(*r2)
```

The signum, or sign function, returns a +1 for positive operands and a –1 for negative operands. The above program assumes that the memory location pointed to by r1 contains a floating-point +1 and that the location pointed to by r2 contains a floating-point –1. The value in the memory location pointed to by r3 is replaced by its signum. (Note: duac[6]=0, therefore Z-field writes are unconditional).

(c) Minimum

```
(1)     a0 = –a1 + a2
(2)     a1 = ifalt(a2)
```

Accumulator a1 is written with the smaller of the two values stored in a1 and a2.

(d) Maximum

```
(1)     a0 = a1 – a2
(2)     a1 = ifalt(a2)
```

Accumulator a1 is written with the larger of the two values in a1 and a2.

(e) Half-Wave Rectification

```
(1)     a1 = *r2
(2)     a1 = ifalt(*r1)    /* r1 points to a floating-point zero */
```

The contents of memory location r2 are half-wave rectified and then stored in a1.

(f) Two-Sided Limiter

```
(1)     a2 = –*r1          /* initialize a2 with negative limit      */
(2)     a1 = –a0 + *r1     /* compare a0 with positive limit         */
(3)     a0 = ifalt(*r1)    /* if a0>limit, replace a0 with limit     */
(4)     a1 = a0 + *r1      /* compare a0 with negative limit         */
(5)     a0 = ifalt(a2)     /* if a0<limit, replace a0 with neg. limit */
```

The two-sided limiter restricts the magnitude of the input. In the example above, it is assumed that the limit is stored in a memory location pointed to by r1. The contents of a0 are limited, and the result is stored back into a0.

4.6 Detailed Description of Instruction Set

This section contains a detailed description of the DSP3210 instruction set presented in sections 4.2 and 4.3. The description consists of a short explanation of the operation performed by the instruction, the flags affected by the instruction, a description of the binary encoding, and an example. The instructions are organized alphabetically by the instruction reference name.

Instruction Reference	Page	Instruction Reference	Page
ADD	4-21	IFALT	4-54
ADD-CARRY REV	4-24	INCR/DECR	4-55
AND	4-25	INT16	4-56
AND-COMPLEMENT	4-26	INT32	4-57
BIT TEST	4-27	IRETURN	4-58
CALL	4-28	LOAD	4-59
COMPARE	4-30	LOAD-IOR	4-61
DO	4-31	NOP	4-62
DSP	4-33	OC	4-63
EXCLUSIVE OR	4-34	OR	4-64
FADD-STORE	4-35	RETURN	4-65
FADD-TAP	4-36	ROTATE LEFT	4-66
FLOAD-STORE	4-37	ROTATE RIGHT	4-67
FLOAT16	4-38	ROUND	4-68
FLOAT32	4-39	SEED	4-69
FMULT-ACC-STORE	4-40	SET	4-70
FMULT-ACC-TAP	4-41	SET24	4-71
FMULT-ADD-STORE	4-42	SFTRST	4-72
FMULT-STORE	4-43	SHIFT LEFT	4-73
FMULT-TAP	4-44	SHIFT-OR	4-75
GOTO	4-45	SHIFT RIGHT	4-76
GOTO-COND	4-47	SHIFT RIGHT-ARITH	4-78
GOTO-LOOP	4-49	STORE	4-81
IC	4-50	STORE-IOR	4-82
IEEE	4-51	SUBTRACT	4-83
IFAEQ	4-52	WAITI	4-85
IFAGT	4-53		

INSTRUCTION:

rD = {[(long)], (short)} rS3 + N (1)
[if(COND)] rD = {[(long)], (short)} rS1 + rS2 (2)
[if(COND)] rD = {[(long)], (short)} rS * 2 (3)
[if(COND)] rD = {[(long)], (short)} rS (4)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzvc

DESCRIPTION

These instructions perform integer add operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. A (short) operation sign extends the result from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected.

(1) is a three-operand add instruction with an immediate operand. The contents of rS are added to a value N, and the result is placed in rD. N is a 16-bit 2's complement number that is MSB-extended to 32 bits. This instruction is useful in computing addresses by the base plus displacement technique.

(2) is a 3-register add instruction. The contents of register rS1 and register rS2 are added, and the result is placed in register rD.

(3) is a multiply by 2 instruction. This instruction multiplies the contents of register rS by two, and the result is placed in register rD. This instruction is performed by adding rS to itself (rS+rS). This instruction should be used for a left shift by 1 when setting of the carry flag is desired.

(4) is an assignment instruction. This instruction assigns the contents of register rS to the register rD. The operation actually performed is rD = rS+r0.

Instruction Replacement Table

Term	Replacement	Meaning
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS, rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer
rS3	r0—r22, pc, pcsh	r0 (hard-wired zero), or one of 22 CAU registers, the program counter, or the program counter shadow register

INSTRUCTION (cont.): rD = {[(long)], (short)} rS3 + N
[if(COND)] rD = {[(long)], (short)} rS1 + rS2
[if(COND)] rD = {[(long)], (short)} rS * 2
[if(COND)] rD = {[(long)], (short)} rS

Replacement Table for CA Instructions (COND) - CAU Conditions		
Condition	**Flag†**	**Meaning**
false	Not applicable	Condition is always false.
true	Not applicable	Condition is always true.
pl	n=0	Result is nonnegative (plus)
mi	n=1	Result is negative (minus)
ne	z=0	Result not equal to zero
eq	z=1	Result equal to zero
vc	v=0	Overflow clear, no overflow
vs	v=1	Overflow set, overflowed
cc	c=0	Carry clear, no carry
cs	c=1	Carry set, carry
ge	n^v=0	Greater than or equal to zero
lt	n^v=1	Less than zero
gt	z | (n^v)=0	Greater than zero
le	z | (n^v)=1	Less than or equal to zero
hi	c | z=0	Greater than (unsigned number)
ls	c | z=1	Less than (unsigned number)

† Symbol interpretation: ^ = XOR; | = OR

Replacement Table for CA Instructions (COND) - DAU Conditions		
Condition	**Flag**	**Meaning**
ane	Z=0	Not equal to zero
aeq	Z=1	Equal to zero
age	N=0	Greater than or equal to zero
alt	N=1	Less than zero
avc	V=0	Overflow clear, no overflow
avs	V=1	Overflow set, overflowed
auc	U=0	Underflow clear, no underflow
aus	U=1	Underflow set, underflowed
agt	N | Z=0	Greater than zero
ale	N | Z=1	Less than or equal to zero

INSTRUCTION (cont.):

rD = {[(long)], (short)} rS3 + N
[if(COND)] rD = {[(long)], (short)} rS1 + rS2
[if(COND)] rD = {[(long)], (short)} rS * 2
[if(COND)] rD = {[(long)], (short)} rS

Replacement Table for CA Instructions (COND) - IO Conditions		
Condition	**Flag**	**Meaning**
ibe	ibf=0	Input buffer empty
ibf	ibf=1	Input buffer full
obe	obe=1	Output buffer empty
obf	obe=0	Output buffer full
syc	sy=0	Sync signal low
sys	sy=1	Sync signal high
fbc	fb=0	Serial frame boundary clear
fbs	fb=1	Serial frame boundary set
ir0s	ir0n=1	IR0N pin is logic one (1).
ir0c	ir0n=0	IR0N pin is logic zero (0).
ir1s	ir1n=1	IR1N pin is logic one (1).
ir1c	ir1n=0	IR1N pin is logic zero (0).

LATENCY

A DAU condition tested by a conditional branch or conditional arithmetic/logic instruction is established by the last DA instruction that affects DAU flags no sooner than four instructions prior to the test. For details on latency effects refer to Secion 4.4.2.

INSTRUCTION FORMAT (1) (Format 5a—b)

Bit	31	30—26	25—21	20—16	15—0
Field	E	0 0 1 0 1	rD	rS3	N

INSTRUCTION FORMAT (2–4) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2, rS3 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (+).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

r11= (short) r1 + 32

Bit	31	30—26	25—21	20—16	15—0
Encoding	0	0 0 1 0 1	0 1 0 1 1	0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
Meaning	(short)		r11	r1	0x0020

ADD-CARRY REV

INSTRUCTION: rD = {[(long)], (short)} rD # N (1)
[if(COND)] rD = {[(long)], (short)} rS1 # rS2 (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzc, v=0

DESCRIPTION

These instructions perform carry reverse ADD operations, i.e., an ADD operation with the carry propagating from MSB to LSB. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. The (short) operation propagates the carry from bit 15 to bit 0 and sign extends the result from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction.

(1) This instruction performs a carry reverse add operation on the contents of register rD with the value N and the result is placed in the register rD. N is a 16-bit 2's complement number MSB-extended to 32-bits.

(2) This instruction performs a carry reverse add operation on the contents of register rS1 with the contents of register rS2, and the result is placed in the register rD. If the instruction is conditionally executed and the COND is true, the result of the operation on rS1 and rS2 is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (See page 4-21 ADD). When rS1 = rS2 (rS), this instruction performs a right shift by one binary place and the LSB of rS is placed in the **c** flag.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (#).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

r2 = r14 # r4

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	0 0 1 1	0 0 0 1 0	0 1 1 1 0	0 0 0 0 0 1	0 0 1 0 0
Meaning	(long)		#	r2	r14	true	r4

INSTRUCTION: rD = {[(long)], (short)} rD & N (1)
[if(COND)] rD = {[(long)], (short)} rS1 & rS2 (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

These instructions perform bit-by-bit, logical AND operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction.

(1) This instruction performs a bit-by-bit, logical AND operation on the contents of register rD with the value N and the result is placed in the register rD. N is a 16-bit 2's complement number that is MSB-extended to 32 bits.

(2) This instruction performs a bit-by-bit, logical AND operation on the contents of register rS1 with the contents of register rS2, and the result is placed in the register rD. If the instruction is conditionally executed and the COND is true, the result of the operation on rS1 and rS2 is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (See page 4-21 ADD).

Instruction Replacement Table

Term	Replacement	Meaning
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (&).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

if(cc) r1 = (short) r1 & r2

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	0	0 0 1 1 0 0	1 1 1 0	0 0 0 0 1	0 0 0 0 1	0 0 1 0 0 0	0 0 0 1 0
Meaning	(short)		&	r1	r1	cc	r2

AND-COMPLEMENT

INSTRUCTION: [if (COND)] rD = {[(long)], (short)} rS1 &~ rS2

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

This instruction performs a bit-by-bit, logical AND operation on the contents of register rS1 with the complement of the contents of register rS2. The result is placed in the register rD. The items in the brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. For the (short) operation, the result is sign extended from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (See page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
F Field: Specifies the ALU function to be performed (&~).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

r5 = (short) r4 &~ r3

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	0	0 0 1 1 0 0	0 1 1 0	0 0 1 0 1	0 0 1 0 0	0 0 0 0 0 1	0 0 0 1 1
Meaning	(short)		&~	r5	r4	(true)	r3

INSTRUCTION: {[(long)], (short)} rD & N (1)
[if(COND)}] {[(long)], (short)} rS1 & rS2 (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

These instructions perform bit-by-bit, logical AND operation, but the results are not stored. These instructions can be used for bit tests. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction.

(1) This instruction performs a bit-by-bit, logical AND operation on the contents of register rD with the value N. N is a 16-bit 2's complement number that is MSB-extended to 32 bits.

(2) This instruction performs a bit-by-bit, logical AND operation on the contents of register rS1 with the contents of register rS2. If the instruction is conditionally executed and the COND is true, the flags are set. If the instruction is conditionally executed and the COND is false, the flags are not affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1,rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (& (bit test)).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

r5 & r13

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	1 1 1 1	0 0 0 0 0	0 0 1 0 1	0 0 0 0 0 1	0 1 1 0 1
Meaning	(long)		& (bit test)		r5	true	r13

CALL

INSTRUCTION: call {N, M, rB, rB+N} (rM)

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction transfers control of the program counter (pc) to a location specified in the braces {}. When this instruction is executed, the value of the program counter (pc) is stored in the specified rM register, and the value in the braces {} is moved to the pc. Register rM contains the address of the instruction that follows the latent instruction. The values in the braces {} specify an address N (16-bit MSB extended to 32-bits), an address M (24-bit zero extended to 32-bits), a CAU register rB, or a value in a CAU register rB plus a displacement value N (16-bit MSB extended to 32-bits). N is always 16 bits and is sign-extended to 32 bits for all operations. M is 24-bits and is zero-extended to 32-bits. When using pc as rB, the syxtax:

pccall label

may be used to perform a pc-relative subroutine call. The linker will compute the appropriate N (offset) value. The call M (rM) instruction is a format 8c instruction. All others are format 4a. **This instruction disables interrupts for one instruction cycle.**

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rB	pc, r0—r22	The program counter, r0 (hard-wired zero), or one of 22 CAU registers
rM	r1—r22	One of 22 CAU registers
N	16-bit number	2's complement integer
M	24-bit number	Unsigned integer

LATENCY

When the above instruction is executed, the instruction immediately following is also executed before the branch occurs. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 4a)

Bit	31—26	25—21	20—16	15—0
Field	0 0 0 1 0 0	rM	rB	N

rM Field: Specifies the register used to store the return address.
rB Field: Specifies the base address register. When N specifies 0, rB is the destination address.
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address. When rB specifies r0, the N field is an absolute address.

INSTRUCTION FORMAT (Format 8c)

Bit	31-29	28-21	20—16	15—0
Field	1 1 1	NE	rM	N

rM Field: Specifies the register used to store the return address.
NE Field: Specifies the most significant 8-bits of the 24-bit integer. NE concatonated with N forms the 24-bit unsigned integer (NE || N).
N Field: Specifies the least significant 16-bits of the 24-bit unsigned integer.

INSTRUCTION (cont.): call {N, M, rB, rB+N} (rM)

EXAMPLE (For field encodings refer to Chapter 10.)

call pc+0x0111 (r20)

Bit	31—26	25—21	20—16	15—0
Encoding	0 0 0 1 0 0	1 1 0 0 0	0 1 1 1 1	0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1
Meaning		r20	pc	0x0111

COMPARE

INSTRUCTION: {[(long)],(short)} rD - N (1)
[if(COND)] {[(long)],(short)} rS1 - rS2 (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzvc

DESCRIPTION

These instructions perform subtract operations, but the results are not stored. These instructions are used in compare operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. A (short) operation sign extends the result from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction.

(1) This instruction subtracts the value N from the contents of register rD. N is a 16-bit, 2's complement number that is MSB-extended to 32 bits.

(2) This instruction subtracts the contents of register rS2 from register rS1. If the instruction is conditionally executed and the COND is true, the flags are set. If the instruction is conditionally executed and the COND is false, the flags are not affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1,rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (- (compare)).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

(long) r1 - 15

Bit	31	30—25	24—21	20—16	15—0
Encoding	1	0 0 1 1 0 1	0 1 1 1	0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
Meaning	(long)		(compare)	r1	15

INSTRUCTION: do K,{L,rM} (1)
dolock K,{L,rM} (2)
doblock {L,rM} (3)

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction performs the next K + 1 instructions L + 1(or rM + 1) times. The argument K is an integer from 0 to 127. The second argument can either be a register rM, or a value from 0 to 2047. If a register rM is specified, only the lower 11 bits of the register are used to specify the loop count. When K = 0, the instruction to be repeated is cached on-chip for the duration of the do loop. This eliminates the repeated fetch of the same instruction.

(1) The do instruction initiates a do loop sequence. DMA cycles and interrupts are supported during do loops. For interrupts, the iteration and instruction count registers are shadowed.

(2) The dolock instruction is used to assert an external control signal, LOCKN, for the duration of the do loop. Interrupts and DMA cycles are disabled during a dolock sequence.

(3) The doblock instruction is used to decompose a block move into quad-word transfers. At the beginning of each quad-word, the BLMN signal is asserted and the MS0—MS3 pins indicate a quad-word transfer size (Note that BLMN and MS0—MS3 are configurable, see Section 6.1 - Bus Interface Signal Descriptions). The only instruction allowed following the *doblock* instruction is *a0=(Z=Y)*a0*, therefore K = 0 is implicit. At the completion of the do block sequence, the DAU flags are undefined. Interrupts and DMAs are disabled during each quad-word transfer. See 6.6.3 Quad-word Block Move Indicators.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
K	0—127	Immediate value from 0 to 127
L	0—2047	Immediate value from 0 to 2047
rM	r1—r22	One of 22 CAU registers

LATENCY

The do, dolock, and doblock instructions have no latency properties. Instructions encompassed by a do loop execute with the same latencies and restrictions therefore interactions between the last instruction of the loop and the first instruction of the loop should be carefully inspected. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 3b)

Bit	31—25	24	23	22—18	17—11	10—0
Field	1000110	B	M		K	L

INSTRUCTION FORMAT (Format 3c)

Bit	31—25	24	23	22—18	17—11	10—5	4—0
Field	1000111	B	M		K		rM

B,M Fields: B specifies the dolock instruction, and M specifies the doblock instruction.
K Field: Specifies the number of instructions to be repeated (K field is encoded with one less than the desired number).
L Field: Specifies the iteration count (L field is encoded with one less than the desired count).
rM Field: Specifies the register containing the iteration count (the register value is one less than the desired iteration count).

INSTRUCTION (cont.): do K,{L,rM}
dolock K,{L,rM}
doblock {L,rM}

EXAMPLE (For field encodings refer to Chapter 10.)

do 4, r20

Bit	31—25	24	23	22—18	17—11	10—5	4—0
Encoding	1 0 0 0 1 1 1	0	0	0 0 0 0 0	0 0 0 0 1 0 0	0 0 0 0 0 0	1 1 0 0 0
Meaning					4		r20

INSTRUCTION: [Z =] aN = dsp(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction converts the Y operand, an IEEE format 32-bit floating-point number, to a DSP3210 format floating-point number, and the result is placed in an accumulator (aN). If the exponent of the IEEE number is 255, it is treated as positive or negative infinity according to its sign. If positive, the converted DSP3210 number is the largest positive representable number in the DSP3210 format, namely 0x7fffffff. If negative, the converted DSP3210 number is the most negative number representable in the DSP3210 format, namely 0x800000ff. If the exponent of the IEEE number is zero, the converted DSP3210 number is 0x00000000, or positive zero. The DSP3210 format does not recognize a negative zero. The denormalized IEEE numbers are not supported. If the IEEE format input operand is NaN (not a number), the NaN error condition is set. If a Z field is specified, the result is also stored to a word address in memory. **Note that the Y operand can not come from an accumulator.**

IEEE-to-DSP Conversion Special Cases			
Y†	aN	Flags Z N U V, NaN	Comments
s=0; exp=255; mant=0	0x7fffffff	0 0 0 1 0	Treated as +∞
s=1; exp=255; mant=0	0x800000ff	0 1 0 0 0	Treated as -∞
s=0; exp=255; mant ≠0	0x7fffffff	0 0 0 1 1	Non-zero mantissa treated as +∞
s=1; exp=255; mant ≠0	0x800000ff	0 1 0 1 1	Non-zero mantissa treated as -∞
s=0; exp= 0; mant ≠0	0x00000000	1 0 1 0 0	Denormalized number treated as zero
s=1; exp= 0; mant ≠0	0x00000000	1 0 1 0 0	Denormalized number treated as zero

† s = sign bit; exp = exponent; mant = mantissa

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5 - Refer to [Z=] aN = ieee(Y))

EXAMPLE (For field encodings refer to Chapter 10.)

*r1++ = a1 = dsp(*r9++)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	1 1 0 1	0 1	0 0 0 0 0 0 0	1 0 0 1 1 1 1	0 0 0 1 1 1 1
Meaning		dsp	a1		*r9++	*r1++

EXCLUSIVE OR

INSTRUCTION: rD = {[(long)],(short)} rD ^ N (1)
[if(COND)] rD = {[(long),(short)]} rS1 ^ rS2 (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz,v=c=0

DESCRIPTION

These instructions perform bit-by-bit, logical XOR (exclusive OR) operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction.

(1) This instruction performs a bit-by-bit, logical XOR operation on the contents of register rD with the value N and the result is placed in the register rD. N is a 16-bit 2's complement number that is MSB-extended to 32 bits.

(2) This instruction performs a bit-by-bit, logical XOR operation on the contents of register rS1 with the contents of register rS2, and the result is placed in the register rD. If the instruction is conditionally executed and the COND is true, the result of the operation on rS1 and rS2 is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table

Term	Replacement	Meaning
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1,rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (^).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

if(ir0c) r2 = r3 ^ r4

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	1 0 0 0	0 0 0 1 0	0 0 0 1 1	1 0 1 1 0 0	0 0 1 0 0
Meaning	(long)		^	r2	r3	ir0c	r4

INSTRUCTION: [Z =] aN = [–] Y {+,–} X

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

In this instruction, the operand specified by the X field is added to or subtracted from the operand or minus the operand specified by the Y field and the result is stored in an accumulator (aN). If a Z field is specified, the result is also be stored to a word address in memory. The operation actually performed is [Z =] aN = [–] Y {+,–} 1.0 * X.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 1)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 0 1	M	0	F	S	N	X	Y	Z

M Field: Specifies a constant value to be used for multiplier input (1.0).
F Field: Specifies sign of operation for adder input.
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X Field: Specifies the multiplier input via accumulator direct or register indirect addressing modes.
Y Field: Specifies the adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a1 = *r5-- - *r7++

Bit	31 30 29	28 27 26	25	24	23	22 21	20—14	13—7	6—0
Encoding	0 0 1	1 0 1	0	0	1	0 1	0 1 1 1 1 1 1	0 1 0 1 1 1 0	0 0 0 0 1 1 1
Meaning		(1.0)		+	–	a1	*r7++	*r5--	No write

FADD-TAP

INSTRUCTION: aN = [–] (Z=Y) {+,–} X

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

In this instruction, the operand specified by the X field is added to or subtracted from the operand specified by the Y field. The result or minus the result is stored in an accumulator (aN). The value of the Y operand is stored to a word address in memory. The operation actually performed is: aN = [–] (Z=Y) {+,–} 1.0 * X.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 4)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 0 1	1 1 0	0	F	S	N	X	Y	Z

F Field: Specifies sign of operation for adder input.
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X Field: Specifies the multiplier input via accumulator direct or register indirect addressing modes.
Y Field: Specifies the adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a1 = (*r13 = a3) - *r1--

Bit	31 30 29	28 27 26	25	24	23	22 21	20—14	13—7	6—0
Encoding	0 0 1	1 1 0	0	0	1	0 1	0 0 0 1 1 1 0	0 0 0 0 0 1 1	1 1 0 1 0 0 0
Meaning		Format 4		+	-	a1	*r1--	a3	*r13

INSTRUCTION: [Z =] aN = [–] Y

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction places the operand or minus the operand specified by the Y field into an accumulator (aN). If a Z field is specified, the result is also be stored to a word address in memory. The operation actually performed is [Z=] aN = [–] Y + 0.0 * a0. If the Y operand is not a valid DSP3210 floating-point number, it will be treated as a diry-zero and the accumulator will be loaded with zero. See Section 3.3.3 32-bit Floating-Point Data Type. To perform memory-to-memory transfers use FMULT-TAP.

Instruction Replacement Table

Replace	Value	Meaning
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 1)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 0 1	M	0	F	S	N	X	Y	Z

M Field: Specifies a constant value to be used for multiplier input (0.0).
F Field: Specifies sign of operation for adder input.
S Field: Specifies sign of operation for product (+).
N Field: Specifies the accumulator used for result.
Y Field: Specifies the adder input via accumulator direct or register indirect addressing modes.
X Field: The default specification is a0 as the multiplier input.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r1 = a2 = *r2--

Bit	31 30 29	28 27 26	25	24	23	22 21	20 -14	13 -7	6- 0
Encoding	0 0 1	1 0 0	0	0	0	1 0	0 0 0 0 0 0 0	0 0 1 0 1 1 0	0 0 0 1 0 0 0
Meaning		(0.0)		+	+	a2	(a0)	*r2--	*r1

FLOAT16

INSTRUCTION: [Z =] aN = float16(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: NZ,V=U=0

DESCRIPTION

This instruction converts the 16-bit 2's complement integer operand specified by the Y field to the equivalent 32-bit floating-point representation. The result is stored in an accumulator (aN). The integer operand Y can specify a 16-bit memory location or an accumulator. When the post-increment or post-decrement addressing mode is used for the Y operand the value of the modifier is +2 or -2, respectively. If a Z field is specified, the result is also stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	16-bit memory location. ++ is +2 and -- is -2. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (float16).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r1++r15 = a3 = float16(*r2)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 0 1 0	1 1	0 0 0 0 0 0 0	0 0 1 0 0 0 0	0 0 0 1 0 0 1
Meaning		float16	a3		*r2	*r1++r15

INSTRUCTION: [Z =] aN = float32(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: NZ,V=U=0

DESCRIPTION

This instruction converts the 32-bit integer operand Y to the equivalent 32-bit floating-point representation. The result is stored in the accumulator aN and may optionally be written to memory. The 32-bit integer operand Y is located in a 32-bit memory location. If a Z field is specified, the result is stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit Field	31—27	26—23	22—21	20—14	13—7	6—0
	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (float32).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r5++r17 = a3 = float32(*r2)

Bit Encoding	31—27	26—23	22 21	20—14	13—7	6—0
	0 1 1 1 1	1 0 1 0	1 1	0 0 0 0 0 0 0	0 0 1 0 0 0 0	0 1 0 1 0 1 1
Meaning		float32	a3		*r2	*r5++r17

FMULT-ACC-STORE

INSTRUCTION: [Z =] aN = [–] aM {+,- } Y*X

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction multiplies the operands specified by the X and Y fields. This product is then added to or subtracted from the contents or minus the contents of an accumulator (aM), and the result is stored in an accumulator (aN). If the Z field is specified, the result is also stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN, aM	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 3)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 1 1	M	0	F	S	N	X	Y	Z

M Field: Specifies the accumulator used for adder input.
F Field: Specifies sign of operation for adder input.
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X,Y Fields: Specifies the multiplier inputs via accumulator direct or register indirect addressingmodes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r1++ = a0 = a1 – *r2-- * *r3++r17

Bit	31 30 29	28 27 26	25	24	23	22 21	20 —14	13 —7	6—0
Encoding	0 1 1	0 0 1	0	0	1	0 0	0 0 1 1 0 1 1	0 0 1 0 1 1 0	0 0 0 1 1 1 1
Meaning		a1		+	–	a0	*r3++r17	*r2--	*r1++

INSTRUCTION: aN = [–] aM {+,–} (Z=Y) * X

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction multiplies the operands specified by the X and Y fields. The result of the multiplication (Y * X) is added to or subtracted from the contents of an accumulator (aM), and the result is stored in an accumulator (aN). The value of the Y field operand is stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN, aM	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 2)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 1 0	M	0	F	S	N	X	Y	Z

M Field: Specifies the accumulator used for adder input.
F Field: Specifies sign of operation for adder input.
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X,Y Fields: Specifies the multiplier inputs via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a0 = - a1 + (*r2-- = *r3++r17) * *r1++

Bit	31 30 29	28 27 26	25	24	23	22 21	20—14	13 —7	6—0
Encoding	0 1 0	0 0 1	0	1	0	0 0	0 0 0 1 1 1 1	0 0 1 1 0 1 1	0 0 1 0 1 1 0
Meaning		a1		–	+	a0	*r1++	*r3++r17	*r2--

FMULT-ADD-STORE

INSTRUCTION: [Z =] aN = [–] Y {+,–} aM * X

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction multiplies the operand specified by the X field with the contents of an accumulator (aM). This product is then added to or subtracted from the operand or minus the operand specified by the Y field and the result is stored in an accumulator (aN). If the Z field is specified, the result is also stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN, aM	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 1)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 0 1	M	0	F	S	N	X	Y	Z

M Field: Specifies the accumulator used for multiplier input.
F Field: Specifies sign of operation for adder input.
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X Field: Specifies the multiplier input via accumulator direct or register indirect addressing modes.
Y Field: Specifies the adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r2++r16 = a2 = *r14++ - a0 * *r10--

Bit	31 30 29	28 27 26	25	24	23	22 21	20—14	13—7	6—0
Encoding	0 0 1	0 0 0	0	0	1	1 0	1 0 1 0 1 1 0	1 1 1 0 1 1 1	0 0 1 0 0 1 0
Meaning		a0		+	–	a2	*r10--	*r14++	*r2++r16

INSTRUCTION: [Z =] aN = [–] Y * X

CLASS: DA-Multiply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction multiplies the operands specified by the X and the Y fields. The product or minus the product is stored in an accumulator (aN). If the Z field is specified, the result is also stored to a word address in memory. The operation actually performed is [Z=] aN = 0.0 {+,–} Y * X

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 3)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 1 1	M	0	F	S	N	X	Y	Z

M Field: Specifies a constant value to be used for adder input (0.0).
F Field: Specifies sign of operation for adder input (+).
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X,Y Fields: Specifies the multiplier inputs via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r2++r17 = a3 = *r12++ * *r11--

Bit	31 30 29	28 27 26	25	24	23	22 21	20—14	13—7	6—0
Encoding	0 1 1	1 0 0	0	0	0	1 1	1 0 1 1 1 1 0	1 1 0 0 1 1 1	0 0 1 0 0 1 1
Meaning		0.0		+	+	a3	*r11--	*r12++	*r2++r17

FMULT-TAP

INSTRUCTION: aN = [–] (Z=Y) * X

CLASS: DA-Mulitply/Accumulate

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction multiplies the operands specified by the X and Y fields. The product Y * X or minus the product is stored in an accumulator (aN). The value of the Y operand is stored to a word address in memory. The operation actually performed is: aN = 0.0 {+,–} (Z=Y) * X. This instruction can be used for 32-bit memory-to-memory transfers of arbitrary data, e.g. a0 = (Z=Y) * a0.

Instruction Replacement Table

Replace	Value	Meaning
aN	a0—a3	One of the four DAU accumulators
X, Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 2)

Bit	31—29	28—26	25	24	23	22—21	20—14	13—7	6—0
Field	0 1 0	M	0	F	S	N	X	Y	Z

M Field: Specifies a constant value to be used for adder input (0.0).
F Field: Specifies sign of operation for adder input (+).
S Field: Specifies sign of operation for product.
N Field: Specifies the accumulator used for result.
X,Y Fields: Specifies the multiplier inputs via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a0 = - (*r2++r16 = *r1) * *r8--

Bit	31 30 29	28 27 26	25	24	23	22 21	20—14	13—7	6—0
Encoding	0 1 0	1 0 0	0	0	1	0 0	1 0 0 0 1 1 0	0 0 0 1 0 0 0	0 0 1 0 0 1 0
Meaning		(0.0)		+	–	a0	*r8--	*r1	*r2++r16

INSTRUCTION: goto {N, M, rB, rB+N, rB+M}

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction is an unconditional goto. When executed, the specified value in the braces {} is moved to the program counter (pc). The values in the braces {} specify either an address N (16-bit MSB extended to 32-bits), an address M (24-bit zero-extended to 32-bits), a CAU register rB, a value in a CAU register rB plus a displacement value N (16-bit MSB extended to 32-bits), or a value in a CAU register rB plus a displacement value M (24-bit zero extended to 32-bits). N is always 16 bits and is sign-extended to 32 bits for all operations. M is always 24-bits and it zero-extended to 32-bits. The goto M instruction is a format 8a instruction. The remaining instructions are encoded as format 0-1b, special cases of the *if (COND) goto* instruction. In this case, the branch is always taken. When using pc as rB, the syxtax:

pcgoto label

may be used to perform a pc-relative branch. The linker will compute the appropriate N (offset) value. **This instruction disables interrupts for one instruction cycle.**

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rB	pc, r0—r22	The program counter, r0 (hard-wired zero), or one of 22 general-purpose registers.
N	16-bit number	2's complement integer
M	24-bit number.	Unsigned integer

LATENCY

When the above instruction is executed, the instruction immediately following is also executed before the branch occurs. Normally, a nop is placed after the branch instruction. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 0—1b)

Bit	31—27	26—21	20—16	15—0
Field	1 0 0 0 0	C	rB	N

C Field: Specifies the condition (true).
rB Field: Specifies the base address register. When N specifies 0, rB is the destination address.
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address. When rB specifies r0, the N field is an absolute address.

INSTRUCTION FORMAT (Format 8a)

Bit	31-29	28-21	20—16	15—0
Field	1 0 1	NE	rB	N

rB Field: Specifies the base address register.
NE Field: Specifies the most significant 8-bits of the 24-bit integer. NE concatonated with N forms the 24-bit unsigned integer (NE || N).
N Field: Specifies the least significant 16-bits of the 24-bit unsigned integer.

GOTO

INSTRUCTION (cont.): goto {N, M, rB, rB+N}

EXAMPLE (For field encodings refer to Chapter 10.)

goto r1

Bit	31—27	26—21	20—16	15—0
Encoding	1 0 0 0 0	0 0 0 0 0 1	0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Meaning		true	r1	0x0000

INSTRUCTION: if(COND) goto {N, rB, rB+N}

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction is a conditional branch instruction that depends on the values of the flags. The condition (CAU COND, DAU COND, or IO COND) is selected from the tables below. CAU COND flags are established by the previous CA instruction that set flags. DAU COND flags are established by the most recent DA instruction that occurs four or more instructions prior to the condition test. IO COND flags can be affected by external events, and program or DMA activity. If the condition evaluates true, the value of the program counter (pc) is replaced with one of the values specified in the braces {}. These values include a CAU register rB, an address N (MSB extended), or a value in a CAU register rB plus a displacement value N (MSB extended). N is a 16-bit 2's complement number and is MSB-extended to 32 bits. When using pc as rB, the syxtax:

if (COND) pcgoto label

may be used to perform a pc-relative branch. The linker will compute the appropriate N (offset) value. **This instruction disables interrupts for one instruction cycle.**

Replacement Table for CA Instructions (COND) - CAU Conditions		
Condition	**Flag†**	**Meaning**
false	Not applicable	Condition is always false
true	Not applicable	Condition is always true
pl	n=0	Result is nonnegative (plus)
mi	n=1	Result is negative (minus)
ne	z=0	Result not equal to zero
eq	z=1	Result equal to zero
vc	v=0	Overflow clear, no overflow
vs	v=1	Overflow set, overflowed
cc	c=0	Carry clear, no carry
cs	c=1	Carry set, carry
ge	n^v=0	Greater than or equal to zero
lt	n^v=1	Less than zero
gt	z | (n^v)=0	Greater than zero
le	z | (n^v)=1	Less than or equal to zero
hi	c | z=0	Greater than (unsigned number)
ls	c | z=1	Less than (unsigned number)

† Symbol interpretation: ^ = XOR; |= OR

Replacement Table for CA Instructions (COND) - DAU Conditions		
Condition	**Flag**	**Meaning**
ane	Z=0	Not equal to zero
aeq	Z=1	Equal to zero
age	N=0	Greater than or equal to zero
alt	N=1	Less than zero
avc	V=0	Overflow clear, no overflow
avs	V=1	Overflow set, overflowed
auc	U=0	Underflow clear, no underflow
aus	U=1	Underflow set, underflowed
agt	N | Z=0	Greater than zero
ale	N | Z=1	Less than or equal to zero

GOTO-COND

INSTRUCTION (cont.): if(COND) goto {N, rB, rB+N}

Replacement Table for CA Instructions (COND) - IO Conditions		
Condition	**Flag**	**Meaning**
ibe	ibf=0	Input buffer empty
ibf	ibf=1	Input buffer full
obe	obe=1	Output buffer empty
obf	obe=0	Output buffer full
syc	sy=0	Sync signal low
sys	sy=1	Sync signal high
fbc	fb=0	Serial frame boundary clear
fbs	fb=1	Serial frame boundary set
ir0s	ir0n=1	IR0N pin is logic one (1).
ir0c	ir0n=0	IR0N pin is logic zero (0).
ir1s	ir1n=1	IR1N pin is logic one (1).
ir1c	ir1n=0	IR1N pin is logic zero (0).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rB	pc, r0—r22	The program counter, r0 (hard-wired zero), or one of 22 general-purpose registers.
N	16-bit number	2's complement integer

LATENCY

When the above instruction is executed, the instruction immediately following is also executed before the branch occurs. A DAU condition tested by a conditional branch or conditional arithmetic/logic instruction is established by the last DA instruction that affects DAU flags no sooner than four instructions prior to the test. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 0-1b)

Bit	31—27	26—21	20—16	15—0
Field	1 0 0 0 0	C	rB	N

C Field: Specifies the condition.
rB Field: Specifies the base address register. When N specifies 0, rB is the destination address.
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address. When rB specifies r0, the N field is an absolute address.

EXAMPLE (For field encodings refer to Chapter 10.)

if(pl) goto r1+0x0800

Bit	31—27	26—21	20—16	15—0
Encoding	1 0 0 0 0	0 0 0 0 1 0	0 0 0 0 1	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
Meaning		pl	r1	0x0800

INSTRUCTION: if(rM– >=0) goto {N, rB, rB+N}

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction is primarily used as a loop counter. The value of rM is treated as a 32-bit number and is decremented each time the instruction is executed. If the value of rM is greater than or equal to zero, the value of the pc changes to one of the specified locations in the braces {}. If rM is less than zero, the next instruction is executed. The values in the braces {} specify a CAU register rB, an address N (MSB extended) or a value in a CAU register rB plus a displacement value N (MSB extended). N is always 16 bits and is sign-extended to 32 bits for all operations. When using pc as rB, the syxtax:

if (rM– >=0) pcgoto label

may be used to perform a pc-relative branch. The linker will compute the appropriate N (offset) value. **This instruction disables interrupts for one instruction cycle.**

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rB	pc, r0—r22	The program counter, r0 (hard-wired zero), or one of 22 general-purpose registers.
rM	r1—r22	One of 22 CAU registers.
N	16-bit number	2's complement integer

LATENCY

When the above instruction is executed, the instruction immediately following is also executed before the branch occurs. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 3a)

Bit	31—26	25—21	20—16	15—0
Field	0 0 0 0 1 1	rM	rB	N

rM Field: Specifies the register used for the loop counter.
rB Field: Specifies the base address register. When N specifies 0, rB is the destination address.
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address. When rB specifies r0, the N field is an absolute address.

EXAMPLE (For field encodings refer to Chapter 10.)

if(r1– >=0) goto r2 - 0x00ab

Bit	31—26	25—21	20—16	15—0
Encoding	0 0 0 0 1 1	0 0 0 0 1	0 0 0 1 0	1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1
Meaning		r1	r2	0xff55

IC

INSTRUCTION: [Z =] aN = ic(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: NZ,V=U=0

DESCRIPTION

This instruction converts the operand specified by the Y field, an 8-bit μ-law, A-law, or unsigned word (depending on the value of the dauc register), to an equivalent 32-bit floating-point number. The result is stored in an accumulator (aN). The Y operand can specify a byte memory location or an accumulator. A detailed description of data formats is presented in 8.2.4 - Type Conversions. If Y specifies an accumulator, the 8-bit word must reside in the most significant byte of the accumulator. If a Z field is specified, the result is also stored to a word address in memory.

Instruction Replacement Table

Replace	Value	Meaning
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	8-bit memory location. ++ is +1 and -- is -1. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit Field	31—27	26—23	22—21	20—14	13—7	6—0
	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (ic).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r1++ = a0 = ic(*r2++)

Bit Encoding	31—27	26—23	22 21	20 —14	13 —7	6—0
	0 1 1 1 1	0 0 0 0	0 0	0 0 0 0 0 0 0	0 0 1 0 1 1 1	0 0 0 1 1 1 1
Meaning		ic	a0		*r2++	*r1++

INSTRUCTION: [Z =] aN = ieee(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction converts the Y operand, a DSP32C format floating-point number, to an IEEE format, floating-point number. The Y operand can specify a memory location or an accumulator. If the exponent of the DSP32C number is zero, the converted IEEE number is 0x00000000 or positive zero. DSP32C to IEEE conversion never results in NaN (not a number). If a Z field is specified, the result is also stored to a word address in memory. In order to get the IEEE number correctly formatted in memory, the Z write must be performed in the instruction that specifies the ieee conversion.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (ieee).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r4-- = a1 = ieee(*r9++)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	1 1 0 0	0 1	0 0 0 0 0 0 0	1 0 0 1 1 1 1	0 1 0 0 1 1 0
Meaning		ieee	a1		*r9++	*r4--

IFAEQ

INSTRUCTION: [Z =] aN = ifaeq(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction stores the Y operand in accumulator aN only if the result of the **most recent** DA instruction is equal to zero. If it is not zero, aN will retain its old value. Regardless, the Y address calculations are performed (register postmodification is performed if specified). The Y operand can specify a word memory location or an accumulator. Based on dauc[6], the Z-field operation can be conditional. If dauc[6]=0, the Z-field operation, if specified, is not conditional. Note that if the test fails, the previous value of aN is stored to memory. If dauc[6] = 1, the write to memory, if specified, is also conditional, but the address postmodifications are always performed. If written, the value of aN is stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (ifaeq).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a0 = ifaeq(*r7)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 1 1 0	0 0	0 0 0 0 0 0 0	0 1 1 1 0 0 0	0 0 0 0 1 1 1
Meaning		ifaeq	a0		*r7	No write

INSTRUCTION: [Z =] aN = ifagt(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction stores the Y operand in accumulator aN if and only if the result of the **most recent** DA instruction is greater than zero. If it is not greater than zero, aN retains its old value. Regardless, the Y address calculations are performed (register postmodification is performed if specified). The Y operand can specify a word memory location or an accumulator. Based on dauc[6], the Z-field operation can be conditional. If dauc[6] = 0, the Z-field operation, if specified, is not conditional. Note that if the test fails, the previous value of aN is stored to memory. If dauc[6] = 1, the write to memory, if specified, is also conditional, but the address postmodifications are always performed. If written, the value of aN is stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit Field	31—27	26—23	22—21	20—14	13—7	6—0
	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (ifagt).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r13 = a1 = ifagt(*r9++r18)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 1 1 1	0 1	0 0 0 0 0 0 0	1 0 0 1 1 0 0	1 1 0 1 0 0 0
Meaning		ifagt	a1		*r9++r18	*r13

IFALT

INSTRUCTION: [Z =] aN = ifalt(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction stores the Y operand in an accumulator (aN) only if the result of the **most recent** DA instruction is negative. If it is not negative, aN retains its old value. Regardless, the Y address calculations are performed (register postmodification is performed if specified). The Y operand can specify a word memory location or an accumulator. Based on dauc[6], the Z-field operation can be conditional. If dauc[6] = 0, the Z-field operation, if specified, is not conditional. Note that if the test fails, the previous value of aN is stored to memory. If dauc[6] = 1, the write to memory, if specified, is also conditional, but the address postmodifications are always performed. If written, the value of aN is stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit Field	31—27	26—23	22—21	20—14	13—7	6—0
	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (ifalt).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r11-- = a2 = ifalt(*r7--)

Bit Encoding	31—27	26—23	22 21	20—14	13—7	6—0
	0 1 1 1 1	0 1 0 1	1 0	0 0 0 0 0 0 0	0 1 1 1 1 1 0	1 0 1 1 1 1 0
Meaning		ifalt	a2		*r7--	*r11--

INSTRUCTION: [if (COND)]rD1 = {[(long)],(short)} rS {+,–} 1 (1)
[if (COND)] sp = [(long)] sp {++,--} (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzvc

DESCRIPTION

(1) This instruction increments or decrements the contents of register rS by 1 and places the result in register rD1 (can not be the stack pointer). The items in the brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. For the (short) operation, the result is sign extended from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

(2) This instruction increments or decrements the contents of the register sp (stack pointer) by 4. The source register and the destination register are sp. The instruction can only be executed as 32-bits. **This instruction disables interrupts for one instruction cycle**. If the instruction is conditionally executed and the COND is true, the result is placed in the register sp, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register sp nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD1	r0—r20,r22	r0 (hard-wired zero), or one of 21 CAU registers
rS	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers

INSTRUCTION FORMAT (1-2) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1,rS2 Fields: Specifies the source register.
F Field: Specifies the ALU function to be performed (+).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

if(vs) r5 = r5 +1

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	0	0 0 1 1 0 0	0 0 0 0	0 0 1 0 1	0 0 1 0 1	0 0 0 1 1 1	1 0 1 1 1
Meaning	(short)		+	r5	r5	vs	(+1)

INT16

INSTRUCTION: [Z =] aN = int16(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction converts the floating-point operand Y to a 16-bit 2's complement integer. The conversion can be done using one of three modes specified by the dauc register bits 4 and 5: round to nearest (dauc[5—4] = 00), truncate towards -∞ (dauc[5—4] = 10), or truncate towards zero (dauc[5—4] = 11). The default (on reset) value of dauc[5—4] = 00. The Y operand can specify an accumulator or a float memory location. The result is stored in the upper 16 bits of an accumulator (aN). The lower portion of aN contains unpredictable data. If the Y operand exceeds the range of allowable 16-bit integers (-2^{15}, $2^{15} - 1$), the result is a saturated 16-bit integer. If a Z field is specified, the result is stored to a half-word address in memory. In order to write the 16-bit result to memory as a 16-bit write, the Z-write must be performed in this instruction. When the post-increment or the post-decrement addressing mode is used for the Z operand, the value of the modifier is +2 or -2, respectively.

Instruction Replacement Table

Replace	Value	Meaning
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	16-bit memory location. ++ is +2 and -- is -2. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (int16).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r3 = a0 = int16(a0)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 0 1 1	0 0	0 0 0 0 0 0 0	0 0 0 0 0 0 0	0 0 1 1 0 0 0
Meaning		int16	a0		a0	*r3

INSTRUCTION: [Z =] aN = int32(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction converts the floating-point operand Y to a 32-bit 2's complement integer. The conversion can be done using one of three modes specified by the dauc register bits 4 and 5: round to nearest (dauc[5—4] = 00), truncate towards -∞ (dauc[5—4] = 10), or truncate towards zero (dauc[5—4] = 11). The default (on reset) value of dauc[5—4] = 00. The Y operand can specify an accumulator or a 32-bit memory location. The result is stored in the 32 MSBs of an accumulator (aN), the mantissa, and guard bits. The remaining bits of aN contain unpredictable data. If the Y operand exceeds the range of allowable 32-bit integers (-2^{31} , $2^{31} - 1$), the result is a saturated 32-bit integer. If a Z field is specified, the result is stored to a word address in memory. In order to get the result correctly formatted in memory, the Z-write must be performed in this instruction.

Instruction Replacement Table

Replace	Value	Meaning
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit Field	31—27	26—23	22—21	20—14	13—7	6—0
	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (int32).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r12++ = a2 = int32(*r1++)

Bit Encoding	31—27	26—23	22 21	20—14	13—7	6—0
	0 1 1 1 1	1 0 1 1	1 0	0 0 0 0 0 0 0	0 0 0 1 1 1 1	1 1 0 0 1 1 1
Meaning		int32	a2		*r1++	*r12++

IRETURN

INSTRUCTION: ireturn

CLASS: CA-Control

CAU FLAGS AFFECTED: Restored from interrupt shadow register

DAU FLAGS AFFECTED: Restored from interrupt shadow register

DESCRIPTION

This instruction performs a return-from-interrupt operation. This operation consists of returning DSP3210 control to the point of interruption and restoring the internal registers from the shadow registers. This instruction is encoded and executes as *if(true) goto pcsh*.

LATENCY

Unlike other CA control group instructions, the ireturn instruction does not have to be followed by a nop, since the instruction following the ireturn in memory is not executed. The instruction that was fetched prior to the interrupt is executed in its place. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Formats 0b and 1b)

Bit	31—27	26—21	20—16	15—0
Field	1 0 0 0 0	C	rB	N

C Field: Specifies the condition (true).
rB Field: Specifies the base address register (pcsh).
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address (zero).

EXAMPLE (For field encodings refer to Chapter 10.)

ireturn

Bit	31—27	26—21	20—16	15—0
Encoding	1 0 0 0 0	0 0 0 0 0 1	1 1 1 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Meaning		true	pcsh	0x0000

INSTRUCTION:

rD = {(char),(byte),(hbyte),(short),(ushort),[(long)]}	MEM	(1)
rD = {(char),(byte),(hbyte),(short),(ushort),[(long)]}	*L	(2)
rD = {(char),(byte),(hbyte),(short),(ushort),[(long)]}	iorS	(3)

CLASS: CA-Data Move

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction moves data from a memory location or IO register specified on the right-hand side to a register rD. The default size of the transfer is (long) and does not have to be specified. The size of the transfer and the CAU register bits affected by the transfer are specified by the key words: (char), (byte), (hbyte), (short), (ushort), and (long). (char), (byte) and (hbyte) are 8-bit transfers. (char) loads bits 7—0 with the data and sign extends bit 7 to bit 31. (byte) loads bits 7—0 with the data and zero extends to bit 31. (hbyte) loads bits 15—8 with the data, and zeros bits 7—0 and 31—16. (short) and (ushort) are 16-bit transfers. (short) loads bits 15—0 with the data and sign extends to bit 31. (ushort) loads bits 15—0 with the data and zero extends to bit 31. (long) is a 32-bit transfer that loads bits 31—0 with the data (see below). **This instruction disables interrupts for one instruction cycle.**

(1) This instruction moves data from a memory location addressed using the register indirect addressing mode to the register rD. Data can be referenced as 8-, 16-, or 32-bits. Post-incrementation or post-decrementation is a function of the data size (8-bit, ±1; 16-bit, ±2; 32-bit, ±4).

(2) This instruction moves data from a memory location addressed using the direct addressing mode (*L) to the register rD. L is a 16-bit unsigned integer that is concatenated with the upper 16-bits representing the location of on-chip memory resources (0x0000 or 0x5003, based on pcw[10]) to form the address.

(3) This instruction moves data from an IO register to the register rD. The size of the selected IO register and the transfer size must be the same.

Destination CAU Register Bits Affected						
Type	**Size**	**31—24**	**23—16**	**15—8**	**7—0**	**Description**
(byte)	8	0	0	0	DATA	Zero-extend to upper 24-bits
(hbyte)	8	0	0	DATA	0	Zero-fill bits 7—0, zero-extend upper 16-bits
(char)	8	S	S	S	DATA	Sign-extend bit 7 to upper 24-bits
(short)	16	S	S	DATA	DATA	Sign-extend bit 15 to upper 16-bits
(ushort)	16	0	0	DATA	DATA	Zero-extend to upper 16-bits
(long)	32	DATA	DATA	DATA	DATA	Load bits 31—0.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
MEM	*rP, *rP++, *rP--, *rP++rI	8-, 16-, or 32-bit memory location. ++ and -- are a function of the data size. rP is a memory pointer (rP = r0—r22); when using the sp (r21), the size of the transfer must be long. rI is an increment register (rI = r0—r22).
iorS	ior0—ior15	One of the IO registers.
L	16-bit number	Unsigned integer - used as a direct address

RESTRICTION

The CAU register loaded and the flags set as a result of the load cannot be referenced in the following instruction (See Section 4.4.1.3, Restriction 3 – CAU and IO Register Load).

LOAD

INSTRUCTION (cont.): rD = {(char),(byte),(hbyte),(short),(ushort),[(long)]} MEM
rD = {(char),(byte),(hbyte),(short),(ushort),[(long)]} *L
rD = {(char),(byte),(hbyte),(short),(ushort),[(long)]} iorS

INSTRUCTION FORMAT (1) (Format 7c)

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	0	T	W	rH	rP	0	0 0 0 0 0	rI

INSTRUCTION FORMAT (2) (Format 7a)

Bit	31	30—26	25	24	23—21	20—16	15—0
Field	0	0 0 1 1 1	0	T	W	rH	L

INSTRUCTION FORMAT (3) (Format 7b)

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	0	T	W	rH	0 0 0 0 0	1	0 0 0 0 0	iorH

T Field: Specifies the direction of the transfer (to a register).
W Field: Specifies the size of the transfer.
rH Field: Specifies the source/destination CAU register (destination).
rP Field: Specifies the memory pointer register.
rI Field: Specifies the increment register.
L Field: Specifies a 16-bit unsigned integer.
iorH Field: Specifies the source/destination IO register (source)

EXAMPLE (For field encodings refer to Chapter 10.)

r10 = (ushort) *0xaa

Bit	31	30—26	25	24	23—21	20—16	15—0
Encoding	0	0 0 1 1 1	0	0	0 1 0	0 1 0 1 0	0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0
Meaning				(to)	(ushort)	r10	0x00aa

INSTRUCTION: iorD = {(byte),(short),[(long)]} MEM

CLASS: CA-Data Move

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction moves data from a memory location to an IO register. The default size of the transfer is (long) and does not have to be specified. The size of the transfer and the IO register bits affected by the transfer are specified by the key words: (byte), (short), and (long). (byte) is an 8-bit transfer that loads bits 7—0 with the data and does not affect bits 8 to bit 31. (short) is a 16-bit transfer that loads bits 15—0 with the data and does not affect bits 16 to 31. (long) is a 32-bit transfer that loads bits 31—0 with the data (see below). **This instruction disables interrupts for one instruction cycle.**

Destination IO Register Bits Affected						
Type	**Size**	**31—24**	**23—16**	**15—8**	**7—0**	**Description**
(byte)	8	X	X	X	DATA	Load bits 7—0 (X means not affected)
(short)	16	X	X	DATA	DATA	Load bits 15—0
(long)	32	DATA	DATA	DATA	DATA	Load bits 31—0

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
MEM	*rP, *rP++, *rP--, *rP++rI	8-, 16-, or 32-bit memory location. ++ and -- are a function of the data size. rP is a memory pointer (rP = r0—r22); when using the sp (r21), the size of the transfer must be long. rI is an increment register (rI = r0—r22).
iorD	ior0—ior15	One of the IO registers

RESTRICTION

The IO register loaded cannot be referenced in the following instruction (See Section 4.4.1.3, Restriction 3 – CAU and IO Register Load). A dauc load from memory must be followed by a CA instruction.

INSTRUCTION FORMAT (Format 7d)

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	1	T	W	iorH	rP	0	0 0 0 0 0	rI

T Field: Specifies the direction of the transfer (to a register).
W Field: Specifies the size of the transfer.
rP Field: Specifies the memory pointer register.
rI Field: Specifies the increment register.
iorH Field: Specifies the source/destination IO register (destination).

EXAMPLE (For field encodings refer to Chapter 10.)

ior8 = (short) *r2

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Encoding	1	0 0 1 1 1	1	0	0 1 1	0 1 0 0 0	0 0 0 1 0	0	0 0 0 0 0	0 0 0 0 0
Meaning				to	(short)	ior8	r2			

NOP

INSTRUCTION: nop

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction is a no-op instruction.This instruction causes execution of the following instruction to be delayed by one processor cycle. This is a special case of the *if(COND) goto* instruction in which the branch is never taken.

INSTRUCTION FORMAT (Formats 0b and 1b)

Bit Field	31—27	26—21	20—16	15—0
	1 0 0 0 0	C	rB	N

C Field: Specifies the condition (false).
rB Field: Specifies the base address register (r0).
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address (zero).

EXAMPLE (For field encodings refer to Chapter 10.)

nop

Bit	31—27	26—21	20—16	15—0
Encoding	1 0 0 0 0	0 0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Meaning		false	r0	0x0000

INSTRUCTION: [Z =] aN = oc(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: None

DESCRIPTION

This instruction converts the floating-point operand specified by the Y field to an 8-bit μ-law, A-law, or unsigned linear value. The data type is specified by dauc register bits 3 and 1: μ-law (dauc[3,1]=0,1), A-law (dauc[3,1] = 0,0), unsigned (dauc[3,1] = 1,x). Conversion to unsigned can be done using one of three modes specified by the dauc register bits 4 and 5: round to nearest (dauc[5—4] = 00), truncate towards -∞ (dauc[5—4] = 10), or truncate towards zero (dauc[5—4] = 11). The default (on reset) value of the dauc register is 0.The Y operand can be a float memory location or an accumulator. The result is stored in the most significant byte of an accumulator (aN). The lower portion of the accumulator is written with unpredictable data. If the value of Y is outside the range of allowable μ-law/A-law/8-bit unsigned numbers, the limiting μ-law/A-law/8-bit unsigned value is returned. For more information see 8.2.4 Type Conversions. No flags are set if this saturation occurs. If a Z field is specified, the result is stored to a byte address in memory. When the post-increment or post-decrement addressing mode is used for the Z operand, the value of the modifier is +1 or –1, respectively. In order to execute the store as 8-bits, the Z-write must be performed in this instruction.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--,*rP++rI	8-bit memory location. ++ is +1 and -- is -1. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (oc).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a1 = oc(a1)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 0 0 1	0 1	0 0 0 0 0 0 0	0 0 0 0 0 0 1	0 0 0 0 1 1 1
Meaning		oc	a1		a1	No write

OR

INSTRUCTION: rD = {[(long)],(short)} rD | N (1)
[if(COND)] rD = {[(long)],(short)} rS1 | rS2 (2)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz,v=c=0

DESCRIPTION

These instructions perform bit-by-bit, logical OR operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction.

(1) This instruction performs a bit-by-bit, logical OR operation on the contents of register rD with the value N and the result is placed in the register rD. N is a 16-bit 2's complement number that is MSB-extended to 32 bits.

(2) This instruction performs a bit-by-bit, logical OR operation on the contents of register rS1 with the contents of register rS2, and the result is placed in the register rD. If the instruction is conditionally executed and the COND is true, the result of the operation on rS1 and rS2 is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1,rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (|).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

r15 = (long) r15 | 0x55

Bit	31	30—25	24—21	20—16	15—0
Encoding	1	0 0 1 1 0 1	1 0 1 0	0 1 1 1 1	0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1
Meaning	(long)		|	r15	0x0055

INSTRUCTION: return (rM)

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction moves the value of rM into the program counter (pc). This is the special case of the unconditional goto instruction where only the CAU registers can be selected. Register rM holds the return address of a subroutine. This instruction is encoded and executes as *if (true)goto rM.* **This instruction disables interrupts for one instruction cycle.**

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rM	r1—r22	One of 22 CAU registers.

LATENCY

When the above instruction is executed, the instruction immediately following is also executed before the branch occurs. For details on latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Formats 0b and 1b)

Bit	31—27	26—21	20—16	15—0
Field	1 0 0 0 0	C	rB	N

C Field: Specifies the condition (true).
rB Field: Specifies the base address register (rM).
N Field: Specifies a 16-bit 2's complement integer used as an offset value for the base address (zero).

EXAMPLE (For field encodings refer to Chapter 10.)

return (r18)

Bit	31—27	26—21	20—16	15—0
Encoding	1 0 0 0 0	0 0 0 0 0 1	1 0 1 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Meaning		true	r18	0x0000

ROTATE LEFT

INSTRUCTION: [if (COND)] rD = {[(long)],(short)} rS <<<1

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzvc

DESCRIPTION

This instruction performs rotate left through carry, i.e., shifts the contents of register rS to the left by one bit and shifts the carry from the previous instruction into bit 0. The result is placed in the register rD. The items in the brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. For the (short) operation, the result is sign extended from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers

NSTRUCTION FORMAT (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS Field: Specifies the source register.
F Field: Specifies the ALU function to be performed (<<<).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

if(cs) r5 = r6 <<<1

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	1 0 1 1	0 0 1 0 1	0 0 1 1 0	0 0 1 0 0 1	0 0 1 1 0
Meaning	(long)		<<<	r5	r6	cs	r6

INSTRUCTION: [if (COND)] rD = {[(long)], (short)} rS >>>1

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzc, v=0

DESCRIPTION

This instruction performs rotate right through carry, i.e., shifts the contents of register rS to the right by one bit and shifts the carry from the previous instruction into the most significant bit (MSB). The result is placed in the register rD. The items in the brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. For the (short) operation, the carry is shifted into bit 15, and the result is sign extended from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers

NSTRUCTION FORMAT (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS Field: Specifies the source register.
F Field: Specifies the ALU function to be performed (>>>).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

if(cs) r5 = r6 >>>1

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	1 0 0 1	0 0 1 0 1	0 0 1 1 0	0 0 1 0 0 1	0 0 1 1 0
Meaning	(long)		>>>	r5	r6	cs	r6

ROUND

INSTRUCTION: [Z =] aN = round(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: NZVU

DESCRIPTION

This instruction rounds the floating-point operand Y to 32-bit floating-point format. It is intended to operate on the contents of an accumulator that contains eight extra bits of precision in the mantissa (although other operands are allowed). The 40-bit operand is rounded to the nearest 32-bit representation and the result is stored in an accumulator (aN). The eight guard bits in the accumulator aN are set to zero. If the 40-bit operand is exactly between two 32-bit values, it is rounded to the greater value. The Y operand can specify an accumulator or a 32-bit memory location. If a Z field is specified, the result is also stored to a word address in memory.

Instruction Replacement Table

Replace	Value	Meaning
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators
Z	*rP, *rP++, *rP--,*rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit	31—27	26—23	22—21	20—14	13—7	6—0
Field	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (round).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder nput via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

*r12++r16 = a3 = round(*r1--)

Bit	31—27	26—23	22 21	20—14	13—7	6—0
Encoding	0 1 1 1 1	0 1 0 0	1 1	0 0 0 0 0 0 0	0 0 0 1 1 1 0	1 1 0 0 0 1 0
Meaning		round	a3		*r1--	*r12++r16

INSTRUCTION: [Z =] aN = seed(Y)

CLASS: DA-Special Function

DAU FLAGS AFFECTED: NZU, V=0

DESCRIPTION

This instruction produces a seed (reciprocal of Y) with mantissa good to three bits of accuracy by inverting all bits of Y, except the sign bit. The floating-point operand Y can specify a float memory location or an accumulator. The result is stored in an accumulator (aN). If a Z field is specified, the result is also stored to a word address in memory.

Instruction Replacement Table		
Replace	**Value**	**Meaning**
aN	a0—a3	One of the four DAU accumulators
Y	*rP, *rP++, *rP--, *rP++rI, a0—a3	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register. One of the four DAU accumulators.
Z	*rP, *rP++, *rP--, *rP++rI	32-bit memory location. ++ is +4 and -- is –4. rP refers to r1—14 and is used as a memory pointer. rI refers to r15—19 and is used as an increment register.

LATENCY

For latency effects refer to Section 4.4.2.

INSTRUCTION FORMAT (Format 5)

Bit Field	31—27	26—23	22—21	20—14	13—7	6—0
	0 1 1 1 1	G	N	0 0 0 0 0 0 0	Y	Z

G Field: Specifies the special function instruction type (seed).
N Field: Specifies the accumulator used for result.
Y Field: Specifies adder input via accumulator direct or register indirect addressing modes.
Z Field: Specifies destination of result via register indirect addressing modes.

EXAMPLE (For field encodings refer to Chapter 10.)

a3 = seed(*r9++)

Bit Encoding	31—27	26—23	22 21	20—14	13—7	6—0
	0 1 1 1 1	1 1 1 0	1 1	0 0 0 0 0 0 0	1 0 0 1 1 1 1	0 0 0 0 1 1 1
Meaning		seed	a3		*r9++	No write

SET

INSTRUCTION: rD = (short) N

CLASS: CA-Data Move

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

This instruction moves the value N into the register rD. N is a 16-bit 2's complement number that is MSB-extended to 32 bits. Flags are set according to 16-bit flag rules (see Table 4-1). The operation actually performed is rD = r0 + N.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (Format 5a)

Bit Field	31—26	25—21	20—16	15—0
	0 0 0 1 0 1	rD	rS	N

rD Field: Specifies the destination register.
rS Field: Specifies the source register (r0).
N Field: Specifies a 16-bit unsigned integer that is sign-extended to 32-bits.

EXAMPLE (For field encodings refer to Chapter 10.)

r2= (short) 0x0800

Bit Encoding Meaning	31—26	25—21	20—16	15—0
	0 0 0 1 0 1	0 0 0 1 0	0 0 0 0 0	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
		r2	(r0)	0x0800

INSTRUCTION: rD = (ushort24) M

CLASS: CA-Data Move

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction moves the value M into register rD. M is a 24-bit unsigned integer that is zero extended to 32-bits.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
M	24-bit number	unsigned integer

INSTRUCTION FORMAT (Format 8b)

Bit	31—29	28—21	20—16	15—0
Field	1 1 0	NE	rD	N

rD Field: Specifies the destination register.
NE Field: Specifies the most significant 8-bits of the 24-bit integer. NE concatonated with N forms the 24-bit unsigned integer (NE || N).
N Field: Specifies the least significant 16-bits of the 24-bit unsigned integer.

EXAMPLE (For field encodings refer to Chapter 10.)

r2= (ushort24) 0x080000

Bit	31—29	28—21	20—16	15—0
Encoding	1 1 0	0 0 0 0 1 0 0 0	0 0 0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Meaning		0x08	r2	0x0000

SFTRST

INSTRUCTION: sftrst

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction performs a soft reset of the processor level. It is intended to be used only from the error level and it returns the processor to the base level. By resetting registers and branching to zero an emulation of reset can be accomplished. This instruction is encoded as spc=(byte)r0.

INSTRUCTION FORMAT (Format 7b) sftrst

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	0	1	0 0 0	0 0 0 0 0	0 0 0 0 0	1	0 0 0 0 0	0 1 0 1 0

INSTRUCTION:

rD = {[(long)], (short)} rD << N (1)
[if(COND)] rD = {[(long)], (short)} rS1 << rS2 (2)
[if(COND)] rD = {[(long)], (short)} rS << 1 (3)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

These instructions perform logical left shift operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. A (long) left shift can be considered an arithmetic left shift, but note that overflow is not detected. (short) indicates that a 16-bit operation is desired. The (short) left shift is a logical left shift and the upper 16 bits of the result are zero. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

(1) This instruction performs a logical left shift on the contents of register rD by N and the result is placed in the register rD. Only the least significant 5 bits of N are used to specify a shift of 0 to 31 binary places.

(2) This instruction performs a logical left shift on the contents of register rS1 by the contents of register rS2, and the result is placed in the register rD. Only the least significant 5 bits of rS2 are used to specify a shift of 0 to 31 binary places.

(3) This instruction performs a conditional logical left shift by one of the register rS and the result is placed in the register rD.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS, rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2-3) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (<<).
C Field: Specifies the condition being tested.

SHIFT LEFT

INSTRUCTION (cont.): rD = {[(long)], (short)} rD << N
[if(COND)] rD = {[(long)], (short)} rS1 << rS2
[if(COND)] rD = {[(long)], (short)} rS << 1

EXAMPLE (For field encodings refer to Chapter 10.)

if(ane) r3 = (short) r2 << 1

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	0	0 0 1 1 0 0	1 1 0 0	0 0 0 1 1	0 0 0 1 0	0 1 0 1 0 0	1 0 1 1 1
Meaning	(short)		<<	r3	r2	ane	(1)

INSTRUCTION: rD = [(long)] rS <<| N

CLASS: CA-Arithmetic Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

This instruction is a shift and OR instruction. The value N is logically shifted to the left by 16 binary places and OR'ed with the contents of rS and the result is stored in the register rD. The instruction is always executed as 32-bits, (long). Specification of (long) is optional. Flags are set according to 32-bit flag rules.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (Format 4b)

Bit	31—26	25—21	20—16	15—0
Field	1 0 0 1 0 0	rD	rS	N

rD Field: Specifies the destination register.
rS Field: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer.

EXAMPLE (For field encodings refer to Chapter 10.)

r2= (long) r1 <<| 0x0800

Bit	31—26	25—21	20—16	15—0
Encoding	1 0 0 1 0 0	0 0 0 1 0	0 0 0 0 1	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
Meaning		r2	r1	0x0800

SHIFT RIGHT

INSTRUCTION:
rD = {[(long)], (short)} rD >> N (1)
[if(COND)] rD = {[(long)], (short)} rS1 >> rS2 (2)
[if(COND)] rD = {[(long)], (short)} rS >> 1 (3)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

These instructions perform logical right shift operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. For the (short) logical right shift, the upper 16 bits of the result are zero. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

(1) This instruction performs a logical right shift on the contents of register rD by N and the result is placed in the register rD. Only the least significant 5 bits of N are used to specify a shift of 0 to 31 binary places.

(2) This instruction performs a logical right shift on the contents of register rS1 by the contents of register rS2, and the result is placed in the register rD. Only the least significant 5 bits of rS2 are used to specify a shift of 0 to 31 binary places.

(3) This instruction performs a conditional logical right shift by one of the register rS and the result is placed in the register rD.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS, rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2-3) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (>>).
C Field: Specifies the condition being tested.

INSTRUCTION (cont.): rD = {[(long)], (short)} rD >> N
[if(COND)] rD = {[(long)], (short)} rS1 >> rS2
[if(COND)] rD = {[(long)], (short)} rS >> 1

EXAMPLE (For field encodings refer to Chapter 10.)

r2 = (long) r2 >> 30

Bit	31	30—25	24—21	20—16	15—0
Encoding	1	0 0 1 1 0 1	1 1 0 0	0 0 0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0
Meaning	(long)		>>	r2	30

SHIFT RIGHT-ARITH

INSTRUCTION: rD = {[(long)], (short)} rD $>> N
[if(COND)] rD = {[(long)], (short)} rS1 $>> rS2
[if(COND)] rD = {[(long)], (short)} rS / 2

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nz, v=c=0

DESCRIPTION

These instructions perform arithmetic right shift operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. For the (short) arithmetic right shift, the result is sign extended from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-21 ADD).

(1) This instruction performs an arithmetic right shift on the contents of register rD by N and the result is placed in the register rD. Only the least significant 5 bits of N are used to specify a shift of 0 to 31 binary places.

(2) This instruction performs an arithmetic right shift on the contents of register rS1 by the contents of register rS2, and the result is placed in the register rD. Only the least significant 5 bits of rS2 are used to specify a shift of 0 to 31 binary places.

(3) This instruction performs a divide-by-two operation; i.e., the arithmetic right shift with sign extension. The contents of register rS are divided by two, and the result is placed in register rD.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS, rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

INSTRUCTION FORMAT (1) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (2-3) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed ($>>).
C Field: Specifies the condition being tested.

INSTRUCTION (cont.): rD = {[(long)], (short)} rD $>> N
[if(COND)] rD = {[(long)], (short)} rS1 $>> rS2
[if(COND)] rD = {[(long)], (short)} rS /2

EXAMPLE (For field encodings refer to Chapter 10.)

r4 = (long) r2 $>> r13

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	1 1 0 1	0 0 1 0 0	0 0 0 1 0	0 0 0 0 0 1	0 1 1 0 1
Meaning	(long)		$>>	r4	r2	true	r13

STORE

INSTRUCTION:

MEM	= {(byte), (hbyte), (short), [(long)]} rS	(1)
*L	= {(byte), (hbyte), (short), [(long)]} rS	(2)
iorD	= {(byte), (hbyte), (short), [(long)]} rS	(3)

CLASS: CA-Data Move

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction moves data from the register rS to a memory location or IO register specified on the left-hand side. The default size of the transfer is (long) and does not have to be specified. The CAU register bits selected and the size of the transfer are specified by the key words: (byte), (hbyte), (short), and (long). (byte) and (hbyte) are 8-bit transfers that select register bits 7—0 and 15—8 respectively. (short) is a 16-bit transfer that selects register bits 15—0. (long) is a 32-bit transfer that selects registers 31—0 (see below).

(1) This instruction moves data from the register rS to a memory location addressed using the register indirect addressing mode. Data can be written as 8-, 16-, or 32-bits. Post-incrementation and post-decrementation is a function of the data size (8-bit, ±1; 16-bit, ±3; 32-bit, ±4).

(2) This instruction moves data from the register rS to a memory location addressed using the direct addressing mode (*L). L is a 16-bit unsigned integer that is concatenated with the upper 16-bits representing the location of on-chip memory resources (0x0000 or 0x5003, based on pcw[10]) to form the address.

(3) This instruction moves data from the register rS to an IO register. The transfer size and the size of the selected IO register must be the same.

Source CAU Register Bits Selected						
Type	**Size**	**31—24**	**23—16**	**15—8**	**7—0**	**Description**
(byte)	8	X	X	X	DATA	Store bits 7—0
(hbyte)	8	X	X	DATA	X	Store bits 15—8
(short)	16	X	X	DATA	DATA	Store bits 15—0
(long)	32	DATA	DATA	DATA	DATA	Store bits 31—0

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rS	r0—r22, pc, pcsh	r0 (hard-wired zero), one of 22 CAU registers, the program counter, or the program counter shadow register
MEM	*rP, *rP++, *rP--, *rP++rI	8-, 16-, or 32-bit memory location. ++ is +1, +2, or +4 for data transfer sizes of 8-, 16-, and 32-bits. -- is -1, -2, and –4 for data transfer sizes of 8-, 16-, and 32-bits. rP is a memory pointer (rP = r0—r22); when using the sp (r21), the size of the transfer must be long. rI is an increment register (rI = r0—r22).
iorD	ior0—ior15	One of the IO registers
L	16-bit number	Unsigned integer - used as a direct address

RESTRICTION

A CAU or IO register store instruction cannot immediately follow a DA instruction (multiply/accumulate or special function) that references memory in the Y field (see Section 4.4.1.2, Restriction 2 – CAU and IO Register Store).

INSTRUCTION (cont.):

MEM = {(byte), (hbyte), (short), [(long)]} rS
*L = {(byte), (hbyte), (short), [(long)]} rS
iorD = {(byte), (hbyte), (short), [(long)]} rS

INSTRUCTION FORMAT (1) (Format 7c)

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	0	T	W	rH	rP	0	0 0 0 0 0	rI

INSTRUCTION FORMAT (2) (Format 7a)

Bit	31	30—26	25	24	23—21	20—16	15—0
Field	0	0 0 1 1 1	0	T	W	rH	L

INSTRUCTION FORMAT (3) (Format 7b)

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	0	T	W	rH	0 0 0 0 0	1	0 0 0 0 0	iorH

T Field: Specifies the direction of the transfer (to).
W Field: Specifies the size of the transfer.
rH Field: Specifies the source/destination register (source).
rP Field: Specifies the memory pointer register.
rI Field: Specifies the increment register.
iorH Field: Specifies the source/destination IO register (destination).
L Field: Specifies a 16-bit unsigned integer.

EXAMPLE (For field encodings refer to Chapter 10.)

*r1++ = (hbyte) r3

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Encoding	1	0 0 1 1 1	0	1	1 0 0	0 0 0 1 1	0 0 0 0 1	0	0 0 0 0 0	1 0 1 1 1
Meaning				from	(hbyte)	r3	r1			rI

STORE-IOR

INSTRUCTION: MEM = {(byte), (short), [(long)]} iorS

CLASS: CA-Data Move

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction moves data from an IO register to a memory location. The default size of the transfer is (long) and does not have to be specified. The size of the transfer and the IO register bits selected by the transfer are specified by the key words: (byte), (short), and (long). (byte) is an 8-bit transfer that selects bits 7—0. (short) is a 16-bit transfer that selects bits 15—0 . (long) is a 32-bit transfer that selects bits 31—0 (see below). Post-incrementation and post-decrementation is a function of the data size (8-bit, ±1; 16-bit, ±2, ; 32-bit, ±4).

Source IO Register Bits Selected						
Type	**Size**	**31—24**	**23—16**	**15—8**	**7—0**	**Description**
(byte)	8	X	X	X	DATA	Select bits 7—0
(short)	16	X	X	DATA	DATA	Select bits 15—0
(long)	32	DATA	DATA	DATA	DATA	Select bits 31—0

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
MEM	*rP, *rP++, *rP--, *rP++rI	8-, 16-, or 32-bit memory location. ++ is +1, +2, or +4 for data transfer sizes of 8-, 16-, and 32-bits. -- is -1, -2, and –4 for data transfer sizes of 8-, 16-, and 32-bits. rP is a memory pointer (rP = r0—r22); when using the sp (r21), the size of the transfer must be long. rI is an increment register (rI = r0—r22).
iorS	ior0—ior15	One of the IO registers

RESTRICTION

A CAU or IO register store instruction cannot immediately follow a DA instruction (multiply/accumulate or special function) that references memory in the Y field (see Section 4.4.1.2, Restriction 2 – CAU and IO Register Store).

INSTRUCTION FORMAT (Format 7d)

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	1	T	W	iorH	rP	0	0 0 0 0 0	rI

T Field: Specifies the direction of the transfer (from a register).
W Field: Specifies the size of the transfer.
rP Field: Specifies the memory pointer register.
rI Field: Specifies the increment register.
iorH Field: Specifies the source/destination IO register (source).

EXAMPLE (For field encodings refer to Chapter 10.)

*r1 = (short) ior0

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Encoding	1	0 0 1 1 1	1	1	1 1 1	0 0 0 0 0	0 0 0 0 1	0	0 0 0 0 0	0 0 0 0 0
Meaning				from	(long)	ior0	r1			

INSTRUCTION:

rD = {[(long)], (short)} rD - N (1)
rD = {[(long)], (short)} N - rD (2)
[if(COND)] rD = {[(long)], (short)} rS1 - rS2 (3)
[if(COND)] rD = {[(long)], (short)} - rS (4)

CLASS: CA-Arithmetic/Logic

CAU FLAGS AFFECTED: nzvc

DESCRIPTION

These instructions perform subtract operations. The items in brackets [ ] are optional. A 32-bit operation, (long), is the default size and does not have to be specified. (short) indicates that a 16-bit operation is desired. A (short) operation sign extends the result from bit 15 to bit 31. Flags are set according to the 16- or 32-bit operation flag rules (see Table 4-1), as specified by the instruction. If the instruction is conditionally executed and the COND is true, the result is placed in the register rD, and the flags are set. If the instruction is conditionally executed and the COND is false, neither the register rD nor the flags are affected. The COND field is the same as for the add operations (see page 4-32).

(1) is a right subtract instruction. This instruction subtracts the value N from the contents of register rD and places the result in the register rD. N is a 16-bit 2's complement number that is MSB-extended to 32 bits.

(2) is a left subtract instruction. This instruction subtracts the contents of register rD from a value N and places the result in the register rD. N is a 2's complement number that is MSB-extended to 32 bits.

(3) is a conditional triadic subtract instruction. This instruction subtracts the contents of register rS2 from register rS1 and places the result in rD.

(4) conditionally changes the sign of the contents of register rS. The result is placed in register rD. This negate instruction sets overflow flag v to one (v = 1) for rS = 0xffff8000 and sets carry flag c to zero (c = 0) for rS = 0x00000000; in all other cases v = 0, c = 1.

Instruction Replacement Table		
Term	**Replacement**	**Meaning**
rD	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
rS, rS1, rS2	r0—r22	r0 (hard-wired zero), or one of 22 CAU registers
N	16-bit number	2's complement integer

SUBTRACT

INSTRUCTION (cont.):
rD = {[(long)], (short)} rD - N
rD = {[(long)], (short)} N - rD
[if(COND)] rD = {[(long)], (short)} rS1 - rS2
[if(COND)] rD = {[(long)], (short)} - rS

INSTRUCTION FORMAT (1-2) (Format 6c—d)

Bit	31	30—25	24—21	20—16	15—0
Field	E	0 0 1 1 0 1	F	rD	N

INSTRUCTION FORMAT (3–4) (Format 6a—b)

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Field	E	0 0 1 1 0 0	F	rD	rS1	C	rS2

E Field: Specifies the size (16/32) of the ALU operation.
rD Field: Specifies the destination register.
rS1, rS2 Fields: Specifies the source register.
N Field: Specifies a 16-bit 2's complement integer that is MSB extended to 32-bits.
F Field: Specifies the ALU function to be performed (-).
C Field: Specifies the condition being tested.

EXAMPLE (For field encodings refer to Chapter 10.)

if(mi) r10 = -r5

Bit	31	30—25	24—21	20—16	15—11	10—5	4—0
Encoding	1	0 0 1 1 0 0	0 1 0 0	0 1 0 1 0	0 0 0 0 0	0 0 0 0 1 1	0 0 1 0 1
Meaning	(long)		—	r10	r0	mi	r5

INSTRUCTION: waiti

CLASS: CA-Control

CAU FLAGS AFFECTED: None

DESCRIPTION

This instruction initiates the wait-for-interrupt mode. The wait-for-interrupt mode is a power-saving mode that stops operations in the CAU and DAU until an interrupt is recognized. If an interrupt is pending when the *waiti* instruction is executed, the *waiti* instruction is ignored. A *nop* instruction should always follow the *waiti* instruction because when the interrupt is recognized this instruction is executed before the interrupt is taken. It should only be executed from the base level. This instruction is encoded as spc=(long)r0.

INSTRUCTION FORMAT (Format 7b) waiti

Bit	31	30—26	25	24	23—21	20—16	15—11	10	9—5	4—0
Field	1	0 0 1 1 1	0	1	1 1 1	0 0 0 0 0	0 0 0 0 0	1	0 0 0 0 0	0 1 0 1 0

Chapter 5

Signal Descriptions

CHAPTER 5. SIGNAL DESCRIPTIONS

5. SIGNAL DESCRIPTION 5-1
Table 5-1 Pin Summary 5-1
Figure 5-1 DSP3210 Signal Groups 5-1
Table 5-2 Pin Descriptions 5-2
Table 5-3 Signal Summary 5-4

5. SIGNAL DESCRIPTIONS

This section contains a brief description of the signals on the DSP3210. Figure 5-1 illustrates the functional organization of these pins which are divided into six categories: memory interface, serial I/O, bit I/O, utility, interrupt, and power supply. Table 5-1 is a summary of the signal pins provided in the 132 PQFP package.

Table 5-1 Pin Summary

Bus Interface	D0—D31, A2—A31, MS0—MS3	66
DRAM Signals	CSN, RW, PBD/BLMN*	3
SRAM Signals	ASN, MRN, MWN	1†
Control	SRDYN, BERRN, AEN, DEN, DLE	5
Arbitration Signals	BRN, BGN, BGACKN, LOCKN	4
Serial I/O	DI, ICK, ILD, DO, OCK, OLD, SY	7
Bit I/O	BIO0—BIO7	8
Utility	CKI, RESTN, ZN	3
Interrupt	IR0N, IR1N, IACK0, IACK1	4
Vss		20
VDD		11
Total		132

* PBD(BLMN) - Page Break Detect and Block Move share the same pin, selection is made via pcw[14—15]
† AEN is multiplexed with MRN and DEN is multiplexed with MWN, selection is made via pcw[12]

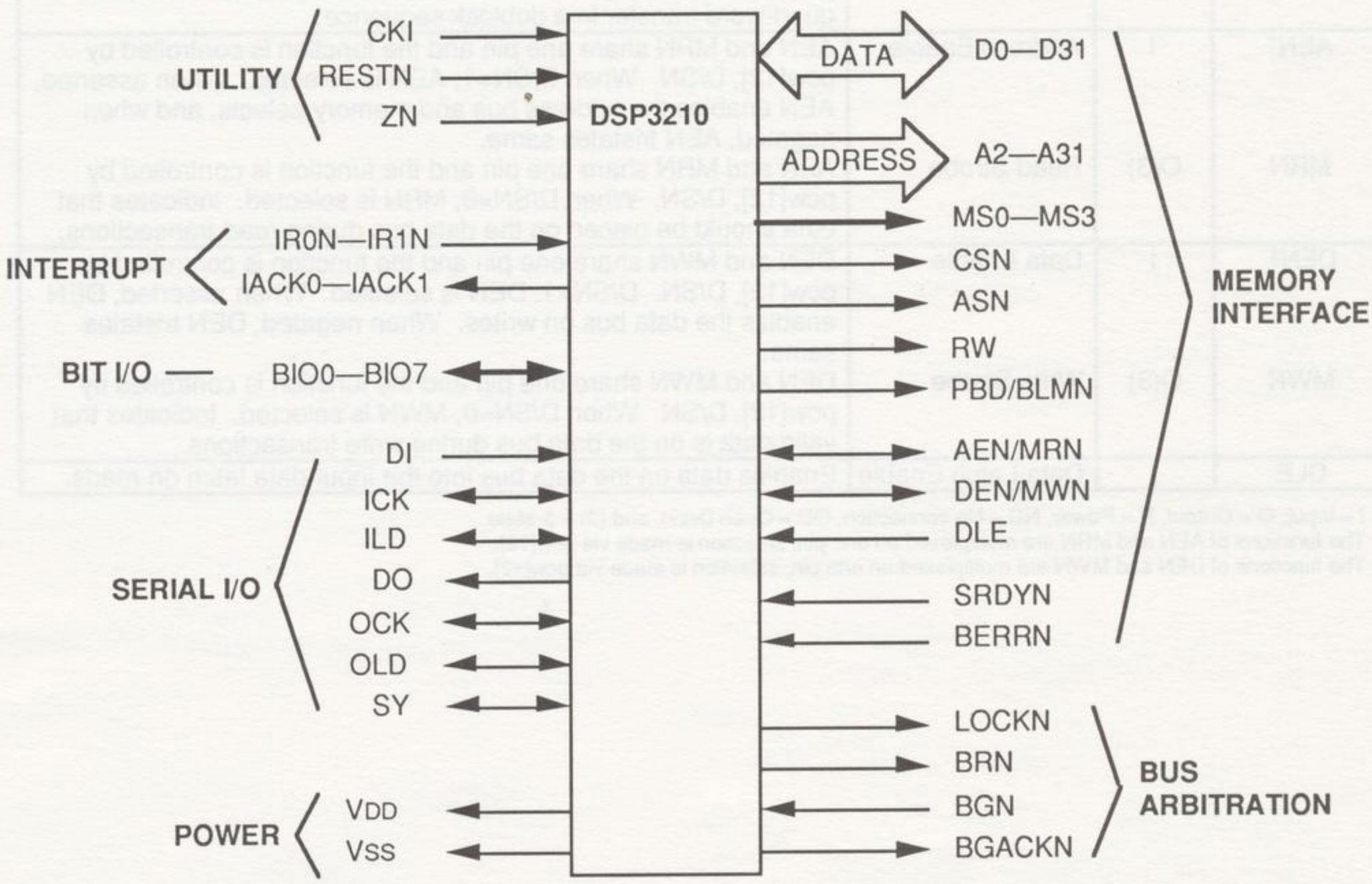


Figure 5-1 DSP3210 Signal Groups

A description of the function of each pin is presented in Table 5-2.

Table 5-2 Pin Descriptions

Pin Name	Type*	Name	Brief Description
D0—D31	I/O(3)	External Memory Data Bus	Bidirectional signals provide the general purpose data path between the DSP3210 and system resources. The data bus transmits and receives 8-, 16-, and 32-bit data.
A2—A31	O(3)	External Memory Address Bus	Provides the word address for DSP bus transfers.
MS0-MS3	O(3)	Memory Selects	Provides memory select information. Signaling based on pcw[9], M/IN: M/IN=1: Memory selects convey data type and word offset. M/IN=0: Memory selects convey byte select information.
CSN	O(3)	Cycle Start	Indicates the beginning of a valid external memory transaction.
ASN	O(3)	Address Strobe	Indicates that address and R/W information are valid.
RW	O(3)	Read/Write	Defines the direction of a data transfer. Signaling based on pcw[11], R/WN: R/WN=1: A high level indicates a read from an external device, a low level indicates a write to an external device. R/WN=0: A low level indicates a read from an external device, a high level indicates a write to an external device.
PBD/BLMN	O(3)	Page Break Detect/Block Move	Provides Page Break Detect or Block Move indication based on pcw[15—14], P/BN. P/BN=01, 1X: Indicates that the current memory access is not on the same page/not sequential as the previous memory access. P/BN=0: Indicates the first word of quad-word transfer in a doblock sequence.
AEN†	I	Address Enable	AEN and MRN share one pin and the function is controlled by pcw[12], D/SN. When D/SN=1, AEN is selected. When asserted, AEN enables the address bus and memory selects, and when negated, AEN tristates same.
MRN	O(3)	Read Strobe	AEN and MRN share one pin and the function is controlled by pcw[12], D/SN. When D/SN=0, MRN is selected. Indicates that data should be placed on the data bus during read transactions.
DEN‡	I	Data Enable	DEN and MWN share one pin and the function is controlled by pcw[12], D/SN. D/SN=1: DEN is selected. When asserted, DEN enables the data bus on writes. When negated, DEN tristates same.
MWN	O(3)	Write Strobe	DEN and MWN share one pin and the function is controlled by pcw[12], D/SN. When D/SN=0, MWN is selected. Indicates that valid data is on the data bus during write transactions.
DLE	I	Data Latch Enable	Enables data on the data bus into the input data latch on reads.

* I = Input, O = Output, P = Power, NC = No connection, OD = Open Drain, and (3) = 3-state.
† The functions of AEN and MRN are multiplexed on one pin, selection is made via pcw[12].
‡ The functions of DEN and MWN are multiplexed on one pin, selection is made via pcw[12].

Table 5-2 Pin Descriptions (continued)

Pin Name	Type*	Name	Brief Description
SRDYN	I	Synchronous Ready	Indicates that a data transfer is complete. When asserted, the DSP3210 completes the current transaction. This is a synchronous input and setup and hold times must be met for reliable operation. SRDYN is used in conjunction with BERRN and BGN to control relinquish, retry, and bus error operations.
BERRN	I	Bus Error	Indicates that a data transfer has encountered a bus error. It is used in conjunction with SRDYN and BGN to control relinquish, retry, and bus error operations.
LOCKN	O(3)	Bus Lock	Asserted when the DSP3210 is requesting interlocked memory transactions.
BRN	O(3)	Bus Request	Asserted when the DSP3210 is requesting mastership of the bus. It indicates that access to external memory is needed.
BGN	I	Bus Grant	Grant for ownership of the bus.
BGACKN	O(3)	Bus Grant Acknowledge	Indicates that the DSP3210 has taken ownership of the data and address busses and control signals. When negated, it indicates that the DSP has placed data, address and control signals in the high-impedance state.
BIO0-BIO7	I/O(3)	Bit I/O	Eight bidirectional signals that can be used for status and control information. Configured by **bioc** register, and data is transferred via **bio** register.
DI	I	Serial Data In	Serial data input.
ICK	I/O(3)	Input Clock	Clock for serial data. In internal mode, ICK is generated on-chip (as specified in **ioc**) and is an output; in external mode, ICK is an input.
ILD	I/O(3)	Input Load	Signal for loading input buffer from internal shift register. In internal mode, ILD is generated on-chip (as specified in **ioc**) and is an output; in external mode, ILD is an input.
DO	O(3)	Serial Data Out	Serial data output.
OCK	I/O(3)	Output Clock	Clock for serial data. In internal mode, OCK is generated on-chip (as specified in **ioc**) and is an output; in external mode, OCK is an input.
OLD	I/O(3)	Output Load	Signal for loading input buffer from internal shift register. In internal mode,OLD is generated on-chip (as specified in **ioc**) and is an output; in external mode, OLD is an input.
SY	I/O(3)	Synchronization	Provides frame synchronization to or from the serial port.
CKI	I	Clock In	System clock.
RESTN	OD	Reset	Highest priority exception. When asserted, the processor enters into the reset state. A low-to-high transition initiates the reset sequence, call r22+0 (r20); nop.
ZN	I	3-state	When asserted, all DSP3210 output pins are 3-stated. ZN has an on-chip pull-up, therefore, it is not necessary that it be connected.
IR0N	I	Interrupt Request 0	Higher priority external interrupt. Maskable in the **emr** register.
IACK0	O(3)	Interrupt Acknowledge 0	Indicates the servicing of the external interrupt request 0.
IR1N	I	Interrupt Request 1	Lower priority external interrupt. Maskable in the **emr** register.
IACK1	O(3)	Interrupt Acknowledge 1	Indicates the servicing of the external interrupt request 1.
V_{DD}	P	Power	
V_{SS}	P	Ground	

* I = Input, O = Output, P = Power, NC = No connection, OD = Open Drain, and (3) = 3-state.

Table 5-3 is a summary of the characteristics of the signals.

Table 5-3 Signal Summary

Signal Function	Signal Name	Input/Output	Active State	3-Stated By			
				ZN	AEN	DEN	BGACKN
Data Bus	D0—D31	Input/Output	High/Low	Yes	—	Yes	—
Address Bus	A2—A31	Output	High/Low	Yes	Yes	—	Yes
Memory Selects	MS0—MS3[1]	Output	High/Low	Yes	Yes	—	Yes
	MS0—MS3[2]	Output	Low	Yes	Yes	—	Yes
Cycle Start	CSN	Output	Low	Yes	—	—	Yes
Address Strobe	ASN	Output	Low	Yes	—	—	Yes
Read/Write	R/W	Output	High/Low	Yes	—	—	Yes
Page Break Detect	PBD	Output	High	Yes	—	—	Yes
Address Enable	AEN[3]	Input	Low	—	—	—	—
Read Strobe	MRN[4]	Output	Low	Yes	—	—	Yes
Data Enable	DEN[5]	Input	Low	—	—	—	—
Write Strobe	MWN[6]	Output	Low	Yes	-	-	Yes
Data Latch Enable	DLE	Input	High	—	—	—	—
Synchronous Ready	SRDYN	Input	Low	—	—	—	—
Bus Error	BERRN	Input	Low	—	—	—	—
Bus Lock	LOCKN	Output	Low	Yes	—	—	Yes
Bus Request	BRN	Output	Low	Yes	—	—	—
Bus Grant	BGN	Input	Low	—	—	—	—
Bus Grant Acknowledge	BGACKN	Output	Low	Yes	—	—	—
Bit I/O	BIO0—BIO7	Input/Output	High/Low	Yes	—	—	—
Serial Data In	DI	Input	High/Low	—	—	—	—
Serial Input Clock	ICK	Input/Output	——	Yes	—	—	—
Serial Input Load	ILD	Input/Output	High–to–Low	Yes	—	—	—
Serial Data Out	DO	Output	High/Low	Yes	—	—	—
Serial Output Clock	OCK	Input/Output	-	Yes	—	—	—
Serial Output Load	OLD	Input/Output	High–to–Low	Yes	—	—	—
Serial Synchronization	SY	Input/Output	High–to–Low	Yes	—	—	—
Clock In	CKI	Input	—	—	—	—	—
Reset	RESTN	Input/Output	Low	—	—	—	—
3-state	ZN	Input	Low	—	—	—	—
Interrupt Request 0	IR0N	Input	Low	—	—	—	—
Interrupt Acknowledge 0	IACK0	Output	High	Yes	—	—	—
Interrupt Request 1	IR1N	Input	Low	—	—	—	—
Interrupt Acknowledge 1	IACK1	Output	High	Yes	—	—	—
Power	VDD	Input	—	—	—	—	—
Ground	Vss	Input	—	—	—	—	—

1. The function of these pins in based on pcw[9]. This entry is for pcw[9] = 0.
2. The function of these pins in based on pcw[9]. This entry is for pcw[9] = 1.
3. The function of this pin in based on pcw[12]. This entry is for pcw[12] = 1.
4. The function of this pin in based on pcw[12]. This entry is for pcw[12] = 0.
5. The function of this pin in based on pcw[12]. This entry is for pcw[12] = 1.
6. The function of this pin in based on pcw[12]. This entry is for pcw[12] = 0.

Chapter 6

Bus Interface

CHAPTER 6. BUS INTERFACE

6 BUS INTERFACE 6-1
6.1 Bus Interface Signal Descriptions 6-1
Figure 6-1 External Memory Select Signaling and Write Data Duplication 6-2
6.2 Processor Control Word 6-4
Figure 6-2 PCW Register Encoding 6-5
6.3 Standard Bus Cycles 6-6
Figure 6-3 Bus Interface State Diagram 6-6
6.3.1 Read Transaction (0-or-more waits) 6-7
6.3.2 Synchronous Write Transaction (0-or-more waits) 6-7
6.3.3 Read Transaction (1 to 3-or-more waits) 6-7
6.3.4 Write Transaction (1 to 3-or-more waits) 6-8
6.3.5 Idle State 6-8
6.4 Bus Error 6-8
Table 6-1 BGN-BERRN-SRDYN Codes and Results 6-8
Figure 6-4 Synchronous Bus Protocol (0-or-more wait-states) 6-9
Figure 6-5 Bus Protocol (1-or-more wait-states) 6-9
Figure 6-6 Bus Protocol (2-or-more wait-states) 6-10
Figure 6-7 Bus Protocol (3-or-more wait-states) 6-10
6.4.1 Bus Error Operation 6-11
Figure 6-8 Read Cycle Retry 6-11
6.5 Bus Arbitration 6-12
6.5.1 Bus Arbitration Operation 6-12
Figure 6-9 Bus Arbitration Example 6-13
6.5.2 DSP3210 BRN Functional Timing 6-13
6.6 Special Interface Capabilities 6-15
6.6.1 Page Break Detect Indicator 6-15
6.6.2 Interlocked Bus Cycle Indicator 6-15
6.6.3 Quad-word Block Move Indicators 6-16
Figure 6-10 Quad-word Block Move Timing 6-17
6.6.4 Multiplexed Address and Data Bus 6-17
Figure 6-11 Multiplexing the Address Bus and Data Bus 6-17
6.6.5 Use of the Data Latch Enable 6-18
Figure 6-12 DLE Timing 6-18

6. BUS INTERFACE

This section provides a functional description of the bus, the signals that control it, and the bus cycles provided for data transfer operations and bus arbitration. For exact timing specifications, refer to Section 12 - Electrical Characteristics.

The DSP3210 allows 8-bit, 16-bit, and 32-bit operands to be located in external memory that is typically 32-bits wide. Section 3.4 Memory Organization describes how data of various sizes may be addressed. The DSP3210 has an address bus that specifies the word address for the transfer and a data bus that transfers the data. Control signals indicate the beginning of the cycle, the data size or bytes of a 32-bit word requested, and the type of transfer, read or write. Bus cycles are synchronous. The ready line, SRDYN, the bus error signal, BERRN, and the bus grant signal, BGN, must meet setup and hold times for reliable operation.

6.1 Bus Interface Signal Descriptions

Name	Description
A2—A31	**Address Bus** (Output; 3-state). The address bus conveys word addresses to the external system. The address space of the DSP3210 is partitioned between on-chip memory and external memory. External addresses initiate bus cycles and are accompanied by control strobes. Internal memory addresses are issued externally, but do not initiate bus cycles, unless 1) an external bys cycle is in progress or 2) the DSP3210 is not granted its bus. Only addresses for external memory are accompanied by control strobes. 3-stated by ZN, AEN, and 3-stated when BGACKN is negated.
D0—D31	**Data Bus** (Bidirectional; 3-state). The external data bus transmits data to and from the DSP3210. The data bus can transmit bytes (8-bit data), halfwords (16-bit data), and words (32-bit data). On write transactions, byte data is replicated on the three remaining bytes of the data bus while halfword data is replicated on the remaining halfword of the data bus. 3-stated by ZN and DEN.
MS0—MS3	**Memory Selects** (Output; 3-state). The information transmitted on MS0—MS3 is controlled by two bits in the PCW register, PCW[9—8]. PCW[9], M/IN, controls whether Motorola or Intel style signaling is used to convey the data type and word offset information. PCW[8], B/LN, controls the byte ordering used when addressing internal or external memory. If M/IN = 1, Motorola semantics are selected and MS0—MS3 have the same meaning as A0-A1 and SIZ0-SIZ1, where MS0 is A0, MS1 is A1, MS2 is SIZ0, and MS3 is SIZ1. If M/IN = 0 and B/LN = 0, Intel semantics are selected and MS0—MS3 have the same meaning as BE0#—BE3#, where MS0 is BE0#, MS1 is BE1#, MS2 is BE2#, and MS3 is BE3#. Figure 6-1 shows the signal information for all data accesses supported by the DSP3210. Also shown is the signaling supplied if M/IN = 0 and B/LN=1. 3-stated by ZN, AEN and 3-stated when BGACKN is negated.
CSN	**Cycle Start** (Output; Active low; 3-state). When asserted (low), this signal indicates the beginning of an external memory transaction. CSN may be used in DRAM memory designs to generate an early RAS signal. 3-stated by ZN and 3-stated when BGACKN is negated.
ASN	**Address Strobe** (Output; Active low; 3-state). When asserted (low), this signal indicates that the address (A2—31) and memory selects (MS0—MS3) are valid. When negated, this signal indicates that the memory transaction is complete. 3-stated by ZN and 3-stated when BGACKN is negated.
RW	**Read/Write** (Output; 3-state). The sense of the read/write signal is controlled by the R/WN bit in the PCW register, PCW[11]. If R/WN = 1, RW has the same semantics as R/W- in a Motorola 680x0. When high, a read transaction is indicated, and when low, a write transaction is indicated. If R/WN = 0, RW has the same semantics as W/R# in an Intel i386. When high, a write transaction is indicated, and when low, a read transaction is indicated. 3-stated by ZN and 3-stated when BGACKN is negated.

Bus Interface Signal Descriptions (continued)

Name	Description
LOCKN	**Bus Lock** (Output; Active Low; 3-state). When asserted (low), this signal indicates that the DSP3210 is requesting interlocked bus transactions. The LOCKN is asserted during the execution of a dolock instruction sequence. 3-stated by ZN and 3-stated when BGACKN is negated.
BLMN	**Block Move** (Output; Active Low; 3-state) BLMN and PBD share one pin. When pcw[15—14]=00, BLMN is selected. When asserted [low], BLMN indicates that the DSP3210 is starting to perform a quad-word block move operation. The BLMN is asserted each time during the **doblock** instruction sequence in which the two least significant word address lines, A2—A3, are 0 and at least four words remain to be transferred. 3-stated by ZN and 3-stated when BGACKN is negated.
PBD	**Page Break Detect** (Output; Active high; 3-state). BLMN and PBD share one pin. When pcw[15—14] = 01, 10, or 11, PBD is selected. When asserted [high], this signal indicates that the current external memory access is not on the same page as the previous external memory access. The page size is specified in the pcw register, pcw[4—6], and the default setting is 256 words. When a page size of 256 words is selected, sequential addressing break detection can also be enabled by pcw[7]. Used to achieve higher speed accesses to static column and page mode DRAMs or VRAMs. PBD is always asserted for the first transaction after the DSP3210 takes ownership of the bus. 3-stated by ZN and 3-stated when BGACKN is negated.

Data Type	Word Offset	Data Type Memory Selects												DSP3210 Write Data			
		M/IN=1 & B/LN=X				M/IN=0 & B/LN=0				M/IN=0 & B/LN=1							
		MS3	MS2	MS1	MS0	MS3	MS2	MS1	MS0	MS3	MS2	MS1	MS0	24—31	16—23	8—15	0—7
8-bit	0	0	1	0	0	1	1	1	0	0	1	1	1	**A**	A	A	A
8-bit	1	0	1	0	1	1	1	0	1	1	0	1	1	B	**B**	B	B
8-bit	2	0	1	1	0	1	0	1	1	1	1	0	1	C	C	**C**	C
8-bit	3	0	1	1	1	0	1	1	1	1	1	1	0	D	D	D	**D**
16-bit	0	1	0	0	0	1	1	0	0	0	0	1	1	**B**	**A**	B	A
16-bit	2	1	0	1	0	0	0	1	1	1	1	0	0	D	C	**D**	**C**
32-bit	0	0	0	0	0	0	0	0	0	0	0	0	0	**D**	**C**	**B**	**A**
4-word	0	1	1	0	0	-†	-	-	-	-	-	-	-	**D**	**C**	**B**	**A**

† 4-word transfers are also signaled with the BLMN pin.

Write Data Key:

A = logical write data D0—7 with little endian byte ordering.
B = logical write data D8—15 with little endian byte ordering.
C = logical write data D16—23 with little endian byte ordering.
D = logical write data D24—31 with little endian byte ordering.

Underline indicates logical position for write data for little endian byte ordering.
Bold indicates logical position for write data for big endian byte ordering.

Figure 6-1 External Memory Select Signaling and Write Data Duplication

Bus Interface Signal Descriptions (continued)

Name	Description
AEN	**Address Enable** (Input; Active Low). AEN and MRN share one pin. When pcw[12] = 1, AEN is selected. When asserted (low), AEN enables the address bus, A2—31, and memory selects, MS0—MS3, and when negated, AEN tristates same (Note that ZN and BGACKN take precedence over AEN). AEN, along with DEN, can be used to multiplex the address bus and data bus.
MRN	**Read Strobe** (Output; Active low; 3-state). AEN and MRN share one pin. When pcw[12] = 0, MRN is selected. MRN is asserted (low) during read transactions to indicate that data may be placed on the data bus (D0—D31). MRN is typically used in SRAM memory designs as an output enable. 3-stated by ZN and 3-stated when BGACKN is negated.
DEN	**Data Enable** (Input; Active Low). DEN and MWN share one pin. When pcw[12] = 1, DEN is selected. When asserted (low), this input enables the data bus (D0—D31) during write transactions (Note that ZN takes precedence over DEN). DEN, along with AEN, can be used to multiplex the address bus and data bus.
MWN	**Write Strobe** (Output; Active low; 3-state). DEN and MWN share one pin. When pcw[12] = 0, MWN is selected. MWN is asserted (low) during write transactions and data setup and hold times are provided around its rising edge. MWN is typically used in SRAM memory designs as a write enable. 3-stated by ZN and 3-stated when BGACKN is negated.
SRDYN	**Synchronous Ready** (Input; Active Low). Indicates that a synchronous data transfer is complete. When asserted, the DSP3210 completes the current transaction. This is a synchronous input and setup and hold times with respect to CKI must be met for reliable operation. SRDYN is used in conjunction with BERRN and BGN to signal bus errors, retry, and relinquish-retry operations.
BERRN	**Bus Error** (Input; Active Low). Indicates that the data transfer is invalid. It is used in conjunction with SRDYN and BGN to indicate that a retry, relinquish/retry, or bus error exception should be taken. A bus error causes the processor to proceed to an exception handler. Refer to Section 6.4 - Bus Error for the encoding and assertion results for SRDYN, BGN and BERRN. This is a synchronous input and setup and hold times with respect to CKI must be met for reliable operation.
DLE	**Data Latch Enable** (Input; Active High). When asserted (high), this input enables data on the data bus (D0—D31) into the input latch. DLE is provided to simplify the design for a DSP3210 communicating on a system bus operating asynchronously from the DSP3210.

6.2 Processor Control Word

The Processor Control Word, pcw, is a 16-bit internal register that can be read or written using selected CA Data Move Instructions, such as pcw = r1. The pcw register is used to configure the external memory wait-states in partitions A and B, the page size for generating the page break detect (PBD), the memory map, the byte ordering, and the signaling supplied for external memory interfacing. Figure 6-2 shows the bit encoding of the pcw register. By writing a one in Bit 13 with a program, the value loaded into the pcw register can be locked so that programs do not inadvertently change its value. The effect of writing this bit is cleared when the processor is reset or when an error exception is taken. pcw[10—13] are loaded with the logic values applied to pins BIO[7:4], respectively. The remaining pcw register bits are set or cleared by reset as shown.

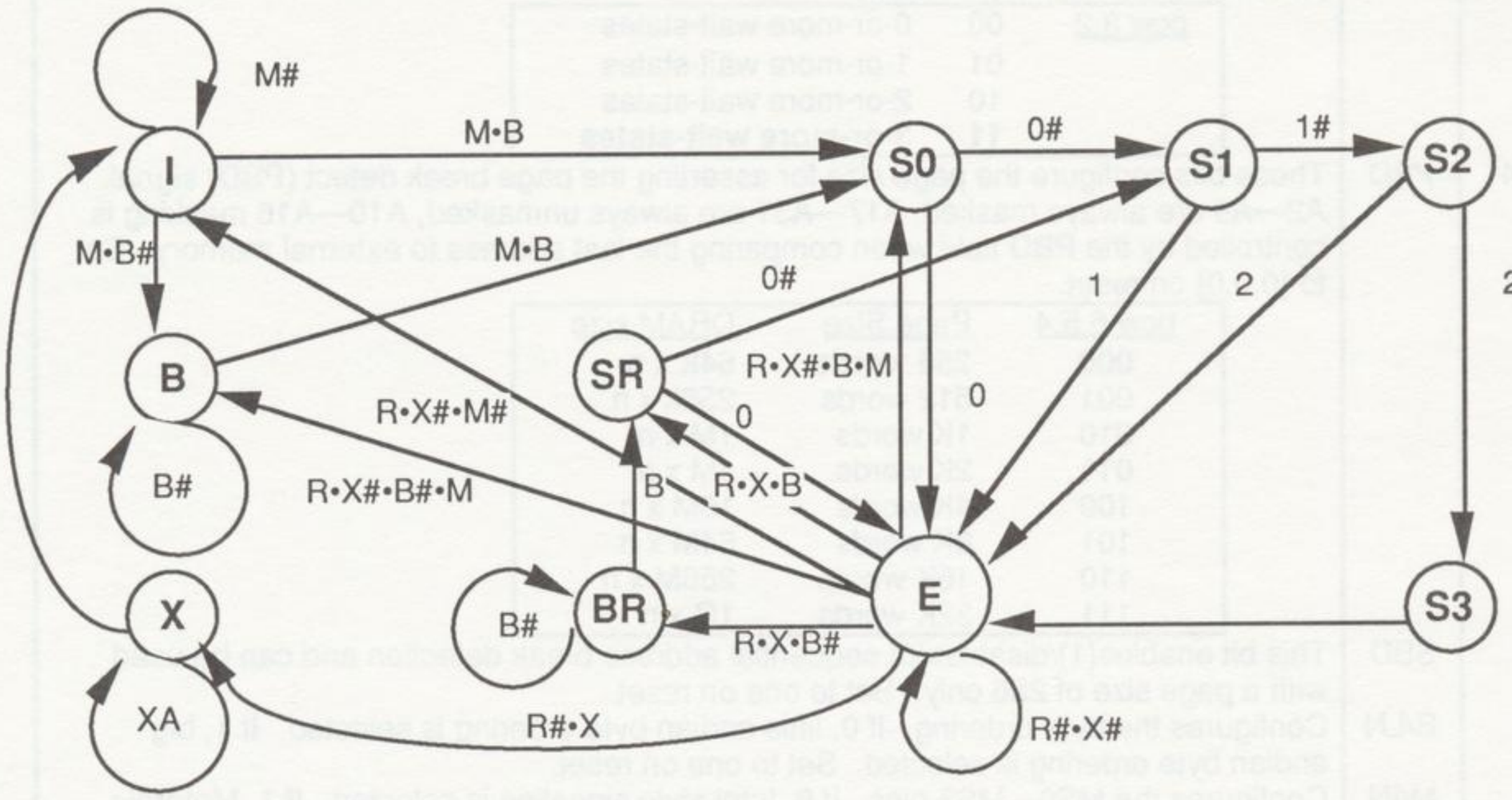
Bit	15 14	13	12	11	10	9	8	7	6 5 4	3 2	1 0
Field	P/BN	BRC	D/SN	R/WN	C/PN	M/IN	B/LN	SBD	PBD	WB	WA

<table>
<tr><th>Bit(s)</th><th>Field</th><th>Function</th></tr>
<tr><td>1,0</td><td>WA</td><td>These bits configure the number of wait-states that are generated for transactions in partition A. Set to {1,1} on reset.
<table>
<tr><td><u>pcw 1,0</u></td><td>00</td><td>0-or-more wait-states</td></tr>
<tr><td></td><td>01</td><td>1-or-more wait-states</td></tr>
<tr><td></td><td>10</td><td>2-or-more wait-states</td></tr>
<tr><td></td><td>11</td><td>3-or-more wait-states</td></tr>
</table></td></tr>
<tr><td>3,2</td><td>WB</td><td>These bits configure the number of wait-states that are generated for transactions in partition B. Set to {1,1} on reset.
<table>
<tr><td><u>pcw 3,2</u></td><td>00</td><td>0-or-more wait-states</td></tr>
<tr><td></td><td>01</td><td>1-or-more wait-states</td></tr>
<tr><td></td><td>10</td><td>2-or-more wait-states</td></tr>
<tr><td></td><td>11</td><td>3-or-more wait-states</td></tr>
</table></td></tr>
<tr><td>6-4</td><td>PBD</td><td>These bits configure the page size for asserting the page break detect (PBD) signal. A2—A9 are always masked, A17—A31 are always unmasked, A10—A16 masking is controlled by the PBD field when comparing the last address to external memory. Set to {0,0,0} on reset.
<table>
<tr><th><u>pcw 6,5,4</u></th><th><u>Page Size</u></th><th><u>DRAM size</u></th></tr>
<tr><td>000</td><td>256 words</td><td>64k x n</td></tr>
<tr><td>001</td><td>512 words</td><td>256k x n</td></tr>
<tr><td>010</td><td>1K words</td><td>1M x n</td></tr>
<tr><td>011</td><td>2K words</td><td>4M x n</td></tr>
<tr><td>100</td><td>4K words</td><td>16M x n</td></tr>
<tr><td>101</td><td>8K words</td><td>64M x n</td></tr>
<tr><td>110</td><td>16K words</td><td>256M x n</td></tr>
<tr><td>111</td><td>32K words</td><td>1G x n</td></tr>
</table></td></tr>
<tr><td>7</td><td>SBD</td><td>This bit enables(1)/disables(0) sequential address break detection and can be used with a page size of 256 only. Set to one on reset.</td></tr>
<tr><td>8</td><td>B/LN</td><td>Configures the byte ordering. If 0, little endian byte ordering is selected. If 1, big endian byte ordering is selected. Set to one on reset.</td></tr>
<tr><td>9</td><td>M/IN</td><td>Configures the MS0—MS3 pins. If 0, Intel style signaling is selected. If 1, Motorola style signaling is selected. Set to one on reset.</td></tr>
<tr><td>10</td><td>C/PN</td><td>Selects between one of two memory maps. If 1, µComputer mode is selected and on-chip memory is at location zero. If 0, µProcessor mode is selected and external memory is at location zero. Reset value based on the state of bio[7] during reset.</td></tr>
<tr><td>11</td><td>R/WN</td><td>Configures the RW pin. If 0, RW pin is active high for write transactions. If 1, RW pin is active high for read transactions. Reset value based on the state of bio[6] during reset.</td></tr>
<tr><td>12</td><td>D/SN</td><td>Configures the operation of pins AEN/MRN and DEN/MWN. If 0, SRAM interface signaling selected (MRN and MWN). If 1, DRAM system bus interface signaling selected (AEN and DEN). Reset value based on the state of bio[5] during reset.</td></tr>
<tr><td>13</td><td>BRC</td><td>Reset value based on the state of bio[4] during reset. Controls Boot ROM routine. Also, when written with a one, modification of the pcw is disabled.</td></tr>
<tr><td>15—14</td><td>P/BN</td><td>Configures the PBD/BLMN signal. Set to {0,0} on reset.
<table>
<tr><td><u>pcw 15,14</u></td><td>00</td><td>BLMN is selected</td></tr>
<tr><td></td><td>01</td><td>PBD is selected; addresses to partition A are monitored.</td></tr>
<tr><td></td><td>10</td><td>PBD is selected; addresses to partition B are monitored.</td></tr>
<tr><td></td><td>11</td><td>PBD is selected; addresses to partitions A and B are monitored.</td></tr>
</table></td></tr>
</table>

Figure 6-2 PCW Register Encoding

6.3 Standard Bus Cycles

The DSP3210 inherently supports synchronous bus cycles. Synchronous bus cycles are completed by handshaking using the SRDYN signal. The point in the transaction at which SRDYN is observed can be altered by configuring the pcw register for zero-, one-, two-, or three-or-more wait-states. The number of wait-states inserted is the longer of the number programmed or the number added using handshaking. The operation of the bus is summarized in Figure 6-3. Sections 6.3.1 to 6.3.5 walk through the operation of the standard bus cycles available in the DSP3210. For the discussion, it assumes that bus error (BERRN) is not asserted and that pcw[9] is configured for SRAM signaling (MRN and MWN).



INPUTS		STATES	
0	Zero programmed wait-states	I	Idle State (Initial State)
1	One programmed wait-state	S0	Initial Bus Cycle State
2	Two programmed wait-states	S1	First Programmed Wait-state
3	Three programmed wait-states	S2	Second Programmed Wait-state
M	External Memory request asserted	S3	Third Programmed Wait-state
B	Bus Grant (BGN) asserted	E	End Cycle State
R	Synchronous Ready (SRDYN) asserted	B	Bus Grant State
X	Bus Error (BERRN) asserted	BR	Bus Relinquish State
XA	Error Exception Acknowledge	SR	Retry
•,+,#	Logical AND,OR, and NOT, respectively	X	Exception State

Figure 6-3 Bus Interface State Diagram

6.3.1 Read Transaction (0-or-more waits)

A read transaction configured with 0-or-more wait-states takes a minimum of two clock periods to complete. Each clock period consists of two phases, ø0 and ø1. Wait-states repeat State E (see Figure 6-4).

State S0 (ø0): The address (A2—A31), the memory selects (MS0—MS3), the read/write signal (RW), and the page break detect (PBD/BLMN) are issued. The cycle start strobe (CSN) is asserted.

State S0 (ø1): The address strobe (ASN), and the read strobe (MRN) are asserted.

State E (ø0): CSN is negated.

State E (ø1): During phase 1, the synchronous ready line (SRDYN) is observed to indicate the end of the transfer. If SRDYN is high, a wait-state is inserted, and if it is low, the transfer is completed. At the end of the transfer, the DSP3210 samples the data bus (D0—D31), and ASN and MRN are negated.

6.3.2 Synchronous Write Transaction (0-or-more waits)

A write transaction configured with 0-or-more wait-states takes a minimum of two clock periods to complete. Each clock period consists of two phases, ø0 and ø1. Wait-states repeat State E (see Figure 6-4).

State S0 (ø0): The address (A2—A31), the memory selects (MS0—MS3), the read/write signal (RW), and the page break detect (PBD/BLMN) are issued. The cycle start strobe (CSN) is asserted.

State S0 (ø1): The address strobe (ASN), and the write strobe (MWN) are asserted.

State E (ø0): The CSN is negated, and data is enabled onto the external data bus (D0—D31).

State E (ø1): MWN is negated. During ø1, the synchronous ready line (SRDYN) is observed to indicate the end of the transfer. If SRDYN is high, a wait-state is inserted, and if it is low, the transfer is completed. At the end of the transfer, the DSP3210 3-states the data bus (D0—D31), and ASN is negated. Note that MWN is negated at the beginning of State E (ø1) even if wait-states are inserted.

6.3.3 Read Transaction (1 to 3-or-more waits)

A read transaction configured with 1 to 3-or-more wait-states takes a minimum of three clock periods. States S2 and S3 are conditionally needed depending on the number of waits selected. External wait-states repeat State E. (see Figures 6-5 to 6-7).

State S0 (ø0): The address (A2—31), the memory selects (MS0—MS3), the read/write signal (RW), and the page break detect (PBD/BLMN) are issued. The cycle start strobe (CSN) is asserted.

State S0 (ø1): The address strobe (ASN), and the read strobe (MRN) are asserted.

State S1 (ø0):

State S1 (ø1):

State S2 (ø0): Entered only if two or three waits are selected .

State S2 (ø1):

State S3 (ø0): Entered only if three waits are selected.

State S3 (ø1):

State E (ø0): CSN is negated.

State E (ø1): During ø1, the synchronous ready line (SRDYN) is observed to indicate the end of the transfer. If SRDYN is high, a wait-state is inserted, and if it is low, the transfer is completed. At the end of the transfer, the DSP3210 samples the data bus (D0—D31), and ASN and MRN are negated.

6.3.4 Write Transaction (1 to 3-or-more waits)

A write transaction configured with 1 to 3-or-more wait-states takes a minimum of three clock periods to complete. States S2 and S3 are conditionally needed depending on the number of waits selected. External waits repeat State E (see Figures 6-5 to 6-7).

State S0 (ø0): The address (A2—31), the memory selects (MS0—MS3), the read/write signal (RW), and the page break detect (PBD) are issued. The cycle start strobe (CSN) is asserted.

State S0 (ø1): The address strobe (ASN), and the write strobe (MWN) are asserted.

State S1 (ø0):

State S1 (ø1):

State S2 (ø0): Entered only if two or three waits are selected. If two or three waits are configured, data is enabled onto the data bus (D0—D31).

State S2 (ø1):

State S3 (ø0): Entered only if three waits are selected.

State S3 (ø1):

State E (ø0): CSN is negated. if one wait is configured, data is enabled onto the external data bus (D0—D31).

State E (ø1): MWN is negated. During phase 1, the synchronous ready line (SRDYN) is observed to indicate the end of the transfer. If SRDYN is high, a wait-state is inserted, and if it is low, the transfer is completed. At the end of the transfer, the DSP3210 3-states the data bus (D0—D31), and ASN is negated. Note that MWN is negated at the beginning of State E (ø1) even if additional wait-states are inserted.

6.3.5 Idle State

An idle bus state is one clock period long. All control strobes, CSN, ASN, MRN, and MWN, are in their negated state. The RW signal signifies a read. The PBD retains its state from the previous bus cycle. The address bus A2—31 will reflect the address of internal memory accesses if they were initiated in the previous processor state. The MS0—MS3 signals will be high.

6.4 Bus Error

The DSP3210 bus protocol expects the assertion of SRDYN to terminate a bus cycle. If the bus cycle encounters an error condition, external circuitry can:

—request the DSP to retry the transaction

—defer the retry by causing the processor to relinquish the bus

—terminate the transaction and cause the processor to enter an exception handler

These actions are signaled via the BGN, BERRN, and SRDYN pins. Table 6-1 shows the action taken by the DSP for all signal conditions.

Table 6-1 BGN-BERRN-SRDYN Codes and Results

BGN	BERRN	SRDYN	Result
0	0	0	Retry bus transaction immediately
1	0	0	Defer retry, relinquish bus immediately
x	0	1	Terminate transaction and take bus error exception.
x	1	0	Normal termination.
x	1	1	Insert wait-state.

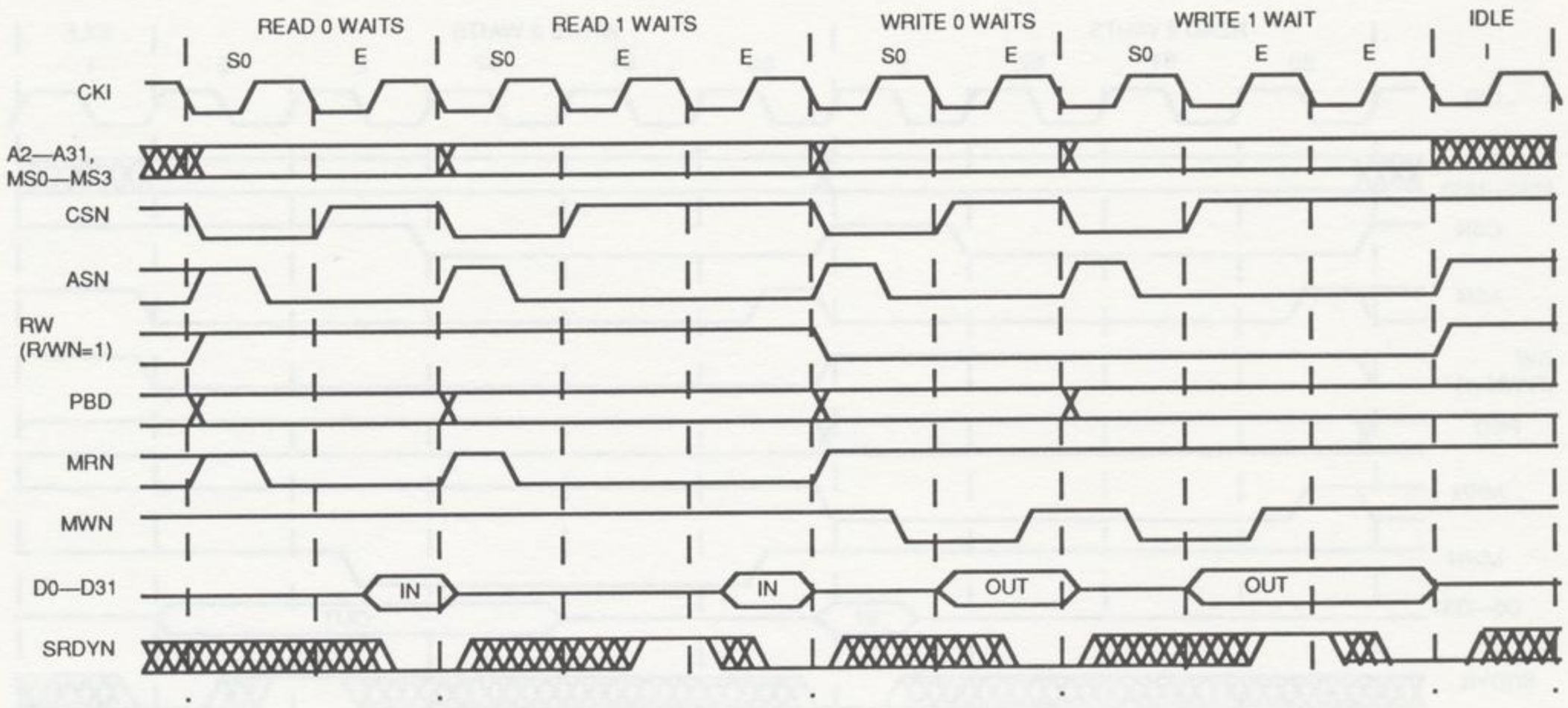


Figure 6-4 Bus Protocol (0-or-more wait-states)

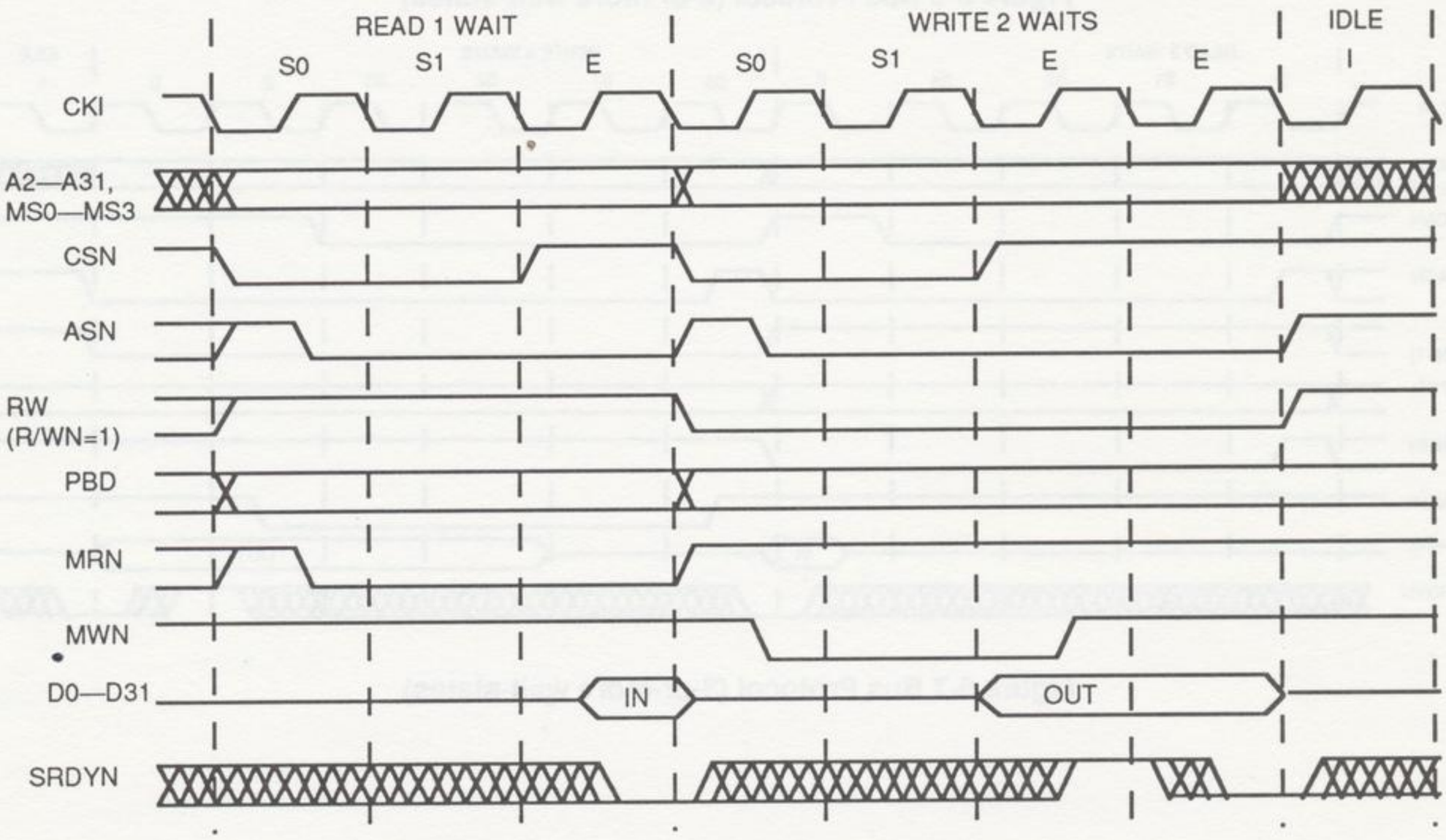


Figure 6-5 Bus Protocol (1-or-more wait-states)

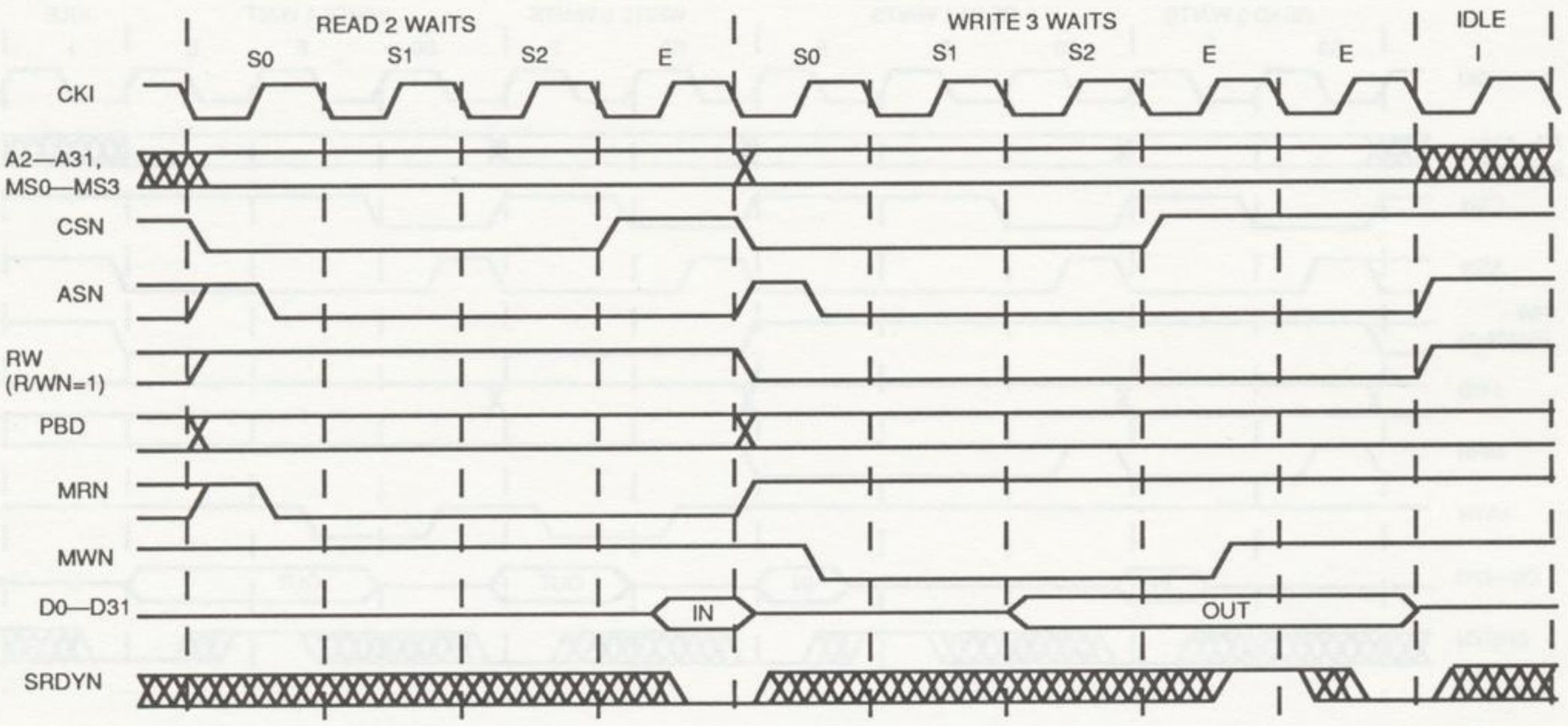


Figure 6-6 Bus Protocol (2-or-more wait-states)

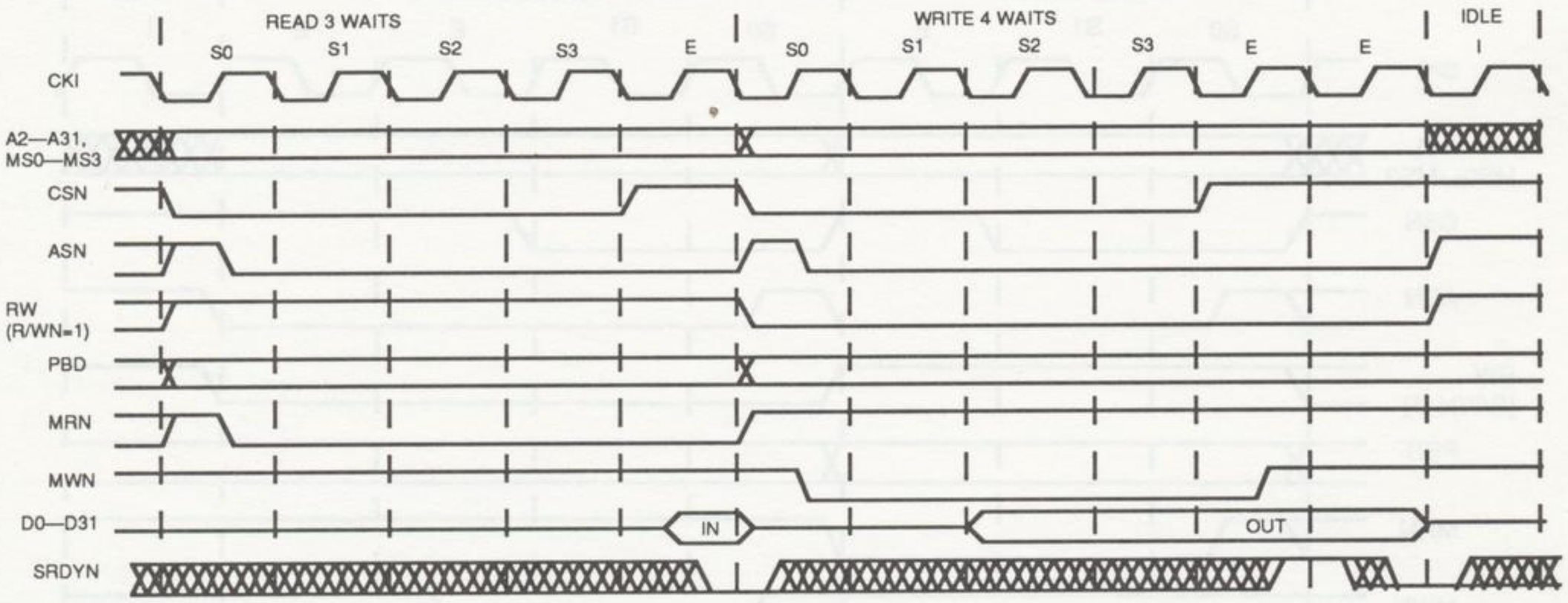


Figure 6-7 Bus Protocol (3-or-more wait-states)

6.4.1 Bus Error Operation

A bus error protocol is signaled to the DSP3210 by the bus error pin, BERRN, in conjunction with SRDYN. The two actions that the DSP can be instructed to take are retry or error exception (stop normal processing and vector to the bus error exception handler, see 7.3 Exception Processing). A retry bus cycle is initiated in the following clock cycle using the same transaction information. Multiple retries can occur if transactions that are retried encounter the retry code when completing. Negating BGN while asserting BERRN and SRDYN provides a relinquish and retry operation where the retry cycle will commence when the DSP regains the bus. When the bus error exception code is detected (BERRN asserted and SRDYN negated), the current bus cycle is terminated, outstanding fetches are invalidated, and the processor proceeds to the exception handler.

Figure 6-8 is a timing diagram of a retry operation.

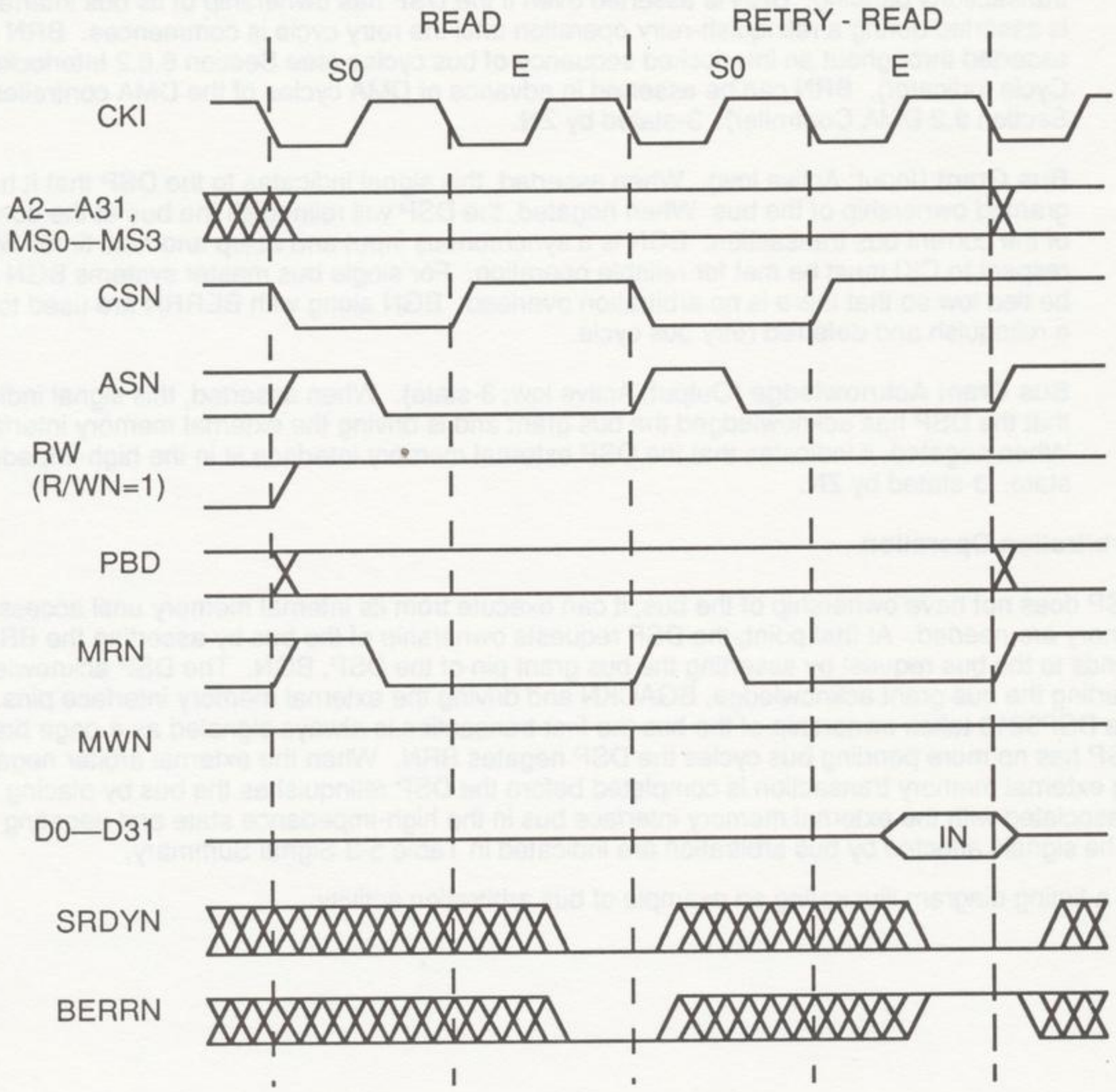


Figure 6-8 Read Cycle Retry

6.5 Bus Arbitration

The DSP3210 shares the external memory interface bus through a request/acknowledge protocol. Bus arbitration is the protocol used by the DSP3210 to become the bus master. An external arbiter controls the arbitration process, and the DSP is a slave device and gains access to the bus by requesting ownership of the bus. The bus arbiter is responsible for assigning priorities to the devices sharing the bus and granting devices the bus.

The three signals associated with the bus arbitration are bus request (BRN), bus grant (BGN), and bus grant acknowledge (BGACKN).

BRN **Bus Request** (Output; Active low; 3-state). When asserted, this signal indicates that the DSP needs to perform an external memory transaction. BRN is negated when there are no further transactions pending. BRN is asserted even if the DSP has ownership of its bus interface. BRN is asserted during a relinquish-retry operation until the retry cycle is commences. BRN is asserted throughout an interlocked sequence of bus cycles (see Section 6.6.2 Interlocked Bus Cycle Indicator). BRN can be asserted in advance of DMA cycles of the DMA controller (see Section 9.2 DMA Controller). 3-stated by ZN.

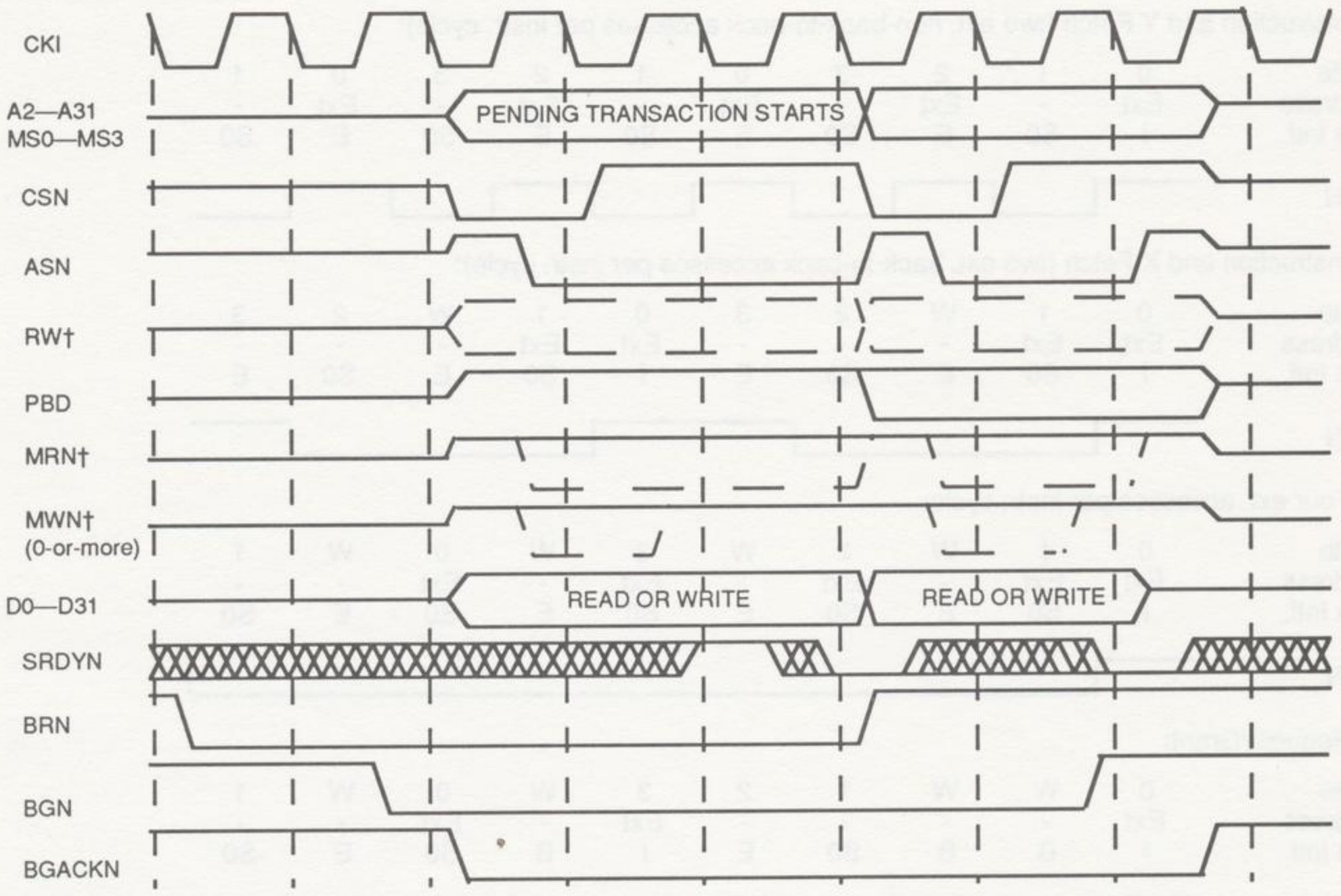
BGN **Bus Grant** (Input; Active low). When asserted, this signal indicates to the DSP that it has been granted ownership of the bus When negated, the DSP will relinquish the bus at the completion of the current bus transaction. BGN is a synchronous input and setup and hold times with respect to CKI must be met for reliable operation. For single bus master systems BGN should be tied low so that there is no arbitration overlead. BGN along with BERRN are used to support a relinquish and deferred retry bus cycle.

BGACKN **Bus Grant Acknowledge** (Output; Active low; 3-state). When asserted, this signal indicates that the DSP has acknowledged the bus grant and is driving the external memory interface. When negated, it indicates that the DSP external memory interface is in the high-impedance state. 3-stated by ZN.

6.5.1 Bus Arbitration Operation

When the DSP does not have ownership of the bus, it can execute from its internal memory until accesses to the external memory are needed. At that point, the DSP requests ownership of the bus by asserting the BRN. The arbiter responds to the bus request by asserting the bus grant pin of the DSP, BGN. The DSP acknowledges the grant by asserting the bus grant acknowledge, BGACKN and driving the external memory interface pins. Note that when the DSP3210 takes ownership of the bus the first transaction is always signaled as a page break. When the DSP has no more pending bus cycles the DSP negates BRN. When the external arbiter negates BGN, any on-going external memory transaction is completed before the DSP relinquishes the bus by placing all output and biputs associated with the external memory interface bus in the high-impedance state and negating BGACKN. The signals affected by bus arbitration are indicated in Table 5-3 Signal Summary.

Figure 6-9 is a timing diagram illustrating an example of bus arbitration activity.



† The signal is dependent on whether it is a read or a write cycle.

Figure 6-9 Bus Arbitration Example

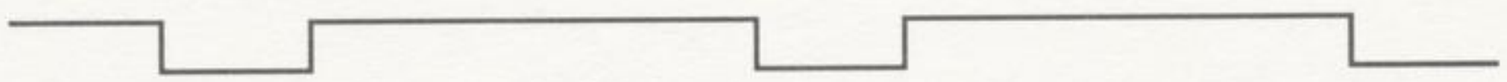
6.5.2 DSP3210 BRN Functional Timing

This section contains examples of BRN functional timing, that is the state of BRN for various sequences of external memory access sequences.

1) Instruction Fetch or doblock (one ext. access per instr. cycle):

State	0	1	2	3	0	1	2	3	0	1
Address	Ext	-	-	-	Ext	-	-	-	Ext	-
Bus Intf.	I	S0	E	I	I	S0	E	I	I	S0

BRN



2) Instruction Fetch or doblock (one ext. access per instr. cycle; one prog. wait):

State	0	1	2	W	3	0	1	2	W	3
Address	Ext	-	-	-	-	Ext	-	-	-	-
Bus Intf.	I	S0	S1	E	I	I	S0	S1	E	I

BRN

3) Instruction and Y Fetch (two ext. non-back-to-back accesses per instr. cycle):

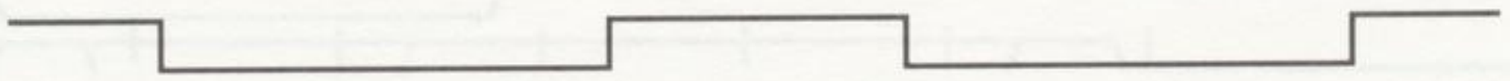
State	0	1	2	3	0	1	2	3	0	1
Address	Ext	-	Ext	-	Ext	-	Ext	-	Ext	-
Bus Intf.	I	S0	E	S0	E	S0	E	S0	E	S0

BRN

4) Instruction and X Fetch (two ext. back-to-back accesses per instr. cycle):

State	0	1	W	2	3	0	1	W	2	3
Address	Ext	Ext	-	-	-	Ext	Ext	-	-	-
Bus Intf.	I	S0	E	S0	E	I	S0	E	S0	E

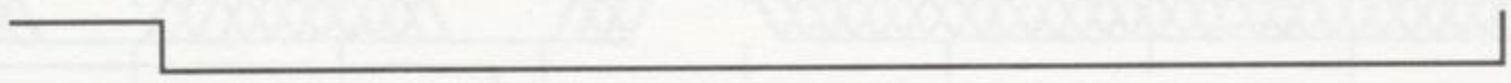
BRN



5) Four ext. accesses per instr. cycle:

State	0	1	W	2	W	3	W	0	W	1
Address	Ext	Ext	-	Ext	-	Ext	-	Ext	-	-
Bus Intf.	I	S0	E	S0	E	S0	E	S0	E	S0

BRN

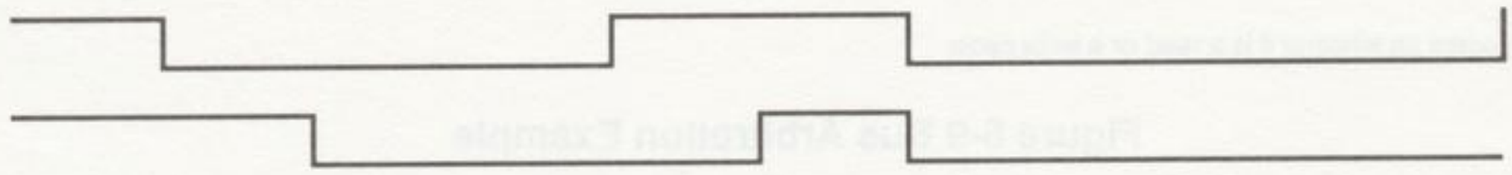


6) Request/Grant:

State	0	W	W	1	2	3	W	0	W	1
Address	Ext	-	-	-	-	Ext	-	Ext	-	-
Bus Intf.	I	B	B	S0	E	I	B	S0	E	S0

BRN

BGN

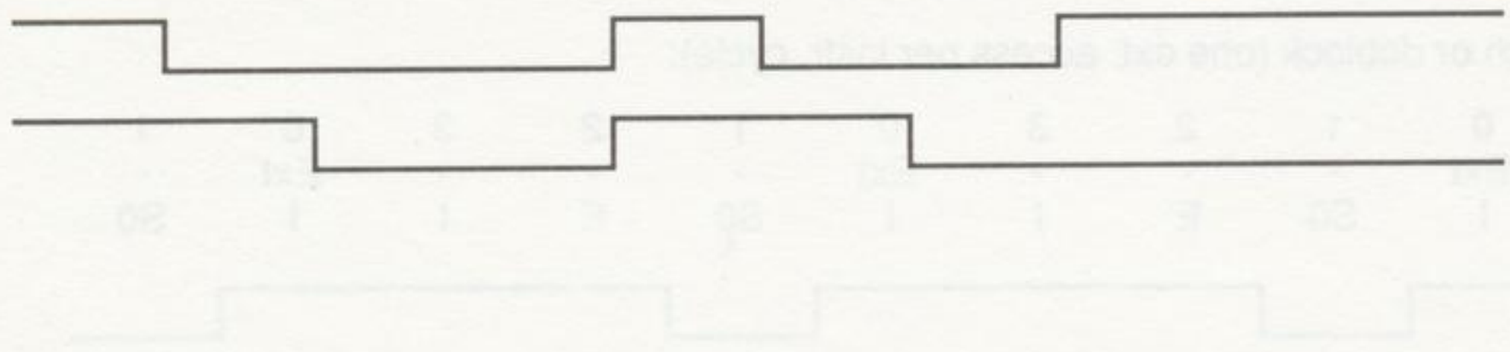


7) Relinquish/Retry:

State	0	W	W	1	2	W	W	W	W	3
Address	Ext	-	-	-	-	-	-	-	-	-
Bus Intf.	I	B	B	S0	E	BR	BR	SR	E	I

BRN

BGN



6.6 Special Interface Capabilities

The DSP3210 has special features that increase system design flexibility. These features improve memory access performance and minimize the number of components necessary when the DSP3210 is being used on an asynchronous and/or multiplexed system bus. The features are implemented with the pins PBD/BLMN, LOCKN, AEN, DEN, and DLE. AEN, DEN, and DLE operate asynchronously with respect to the rest of the external memory interface. Descriptions of these signals is provided in Section 6-1 Bus Interface Signal Descriptions.

6.6.1 Page Break Detect Indicator

The page break detect signal can be used to increase system performance in systems using DRAMs or VRAMs. Bits 4—7 and 14—15 of the pcw register are used to configure the page break detect signal. Bits 4—6 define the page size of the memory device. See Figure 6-2 PCW Register for page size settings. Page sizes can be selected that correspond to DRAMs from 64K to 1G and VRAMs with a 256 bit shift register.

Bits 14—15 define the address space for which page break detection is desired. Partition A, Partition B, or the entire external memory space can be selected as the region the page break detect is monitoring. Addresses outside the selected region are ignored and have no effect on the page break detect signal. This is useful when program and data are separated in different physical external memory devices and increased performance can be achieved by monitoring accesses to only one of the partitions. The page break pin is asserted when an access to the monitored external memory space is not to the same page as the previous access to the monitored external memory space.

Bit 7 is used when the page size is 256 words to enable detection of sequential addressing breaks. This is useful when using VRAMs where data must be read/written sequentially in order to maximize performance. If the next address is not the next word address, $A_{(N+1)} \neq A_{(N)} + 4$, then a sequential addressing break is detected and the page break pin is asserted.

On reset, the page break detect pin is configured to convey BLMN information. This signal will be negated (logic high) which also can be viewed as signaling a page break until the pcw register is set with the appropriate value. The page break is always asserted when the DSP3210 regains bus mastership even if the access is to the same page as the previous external access. Similarly, the page break detect is asserted on the retry of a relinquish-retry operation.

6.6.2 Interlocked Bus Cycle Indicator

The DSP3210 provides a signal, LOCKN, to indicate that it expects to maintain ownership of the bus for one or more additional bus transactions. The LOCKN signal is asserted during the execution of a **dolock** instruction sequence. The bus request pin, BRN, is also asserted for the duration of the dolock instruction sequence. This can be used to perform secure semaphore updates in shared memory systems by emulating a read-modify-write operation within the **dolock** sequence. These instruction sequences are typically not repeated but the structure does allow for it. Interrupts and DMA are disabled during the dolock sequence.

Up to 128 in-line instructions can be encompassed by the dolock indicators, therefore complex and arbitrary bus interlocked sequences can be performed at an overhead of one instruction cycle to execute the dolock instruction. The LOCKN indicator is asserted during the Operand Fetch cycle of the instruction pipeline (see Figure 7-1A Pipeline for a Single DA Instruction). It is not intended that DA instructions with Z-field writes be used in interlocked sequences because the LOCKN indicator will be negated before the write is performed, but four nops following the last DAU write will accomodate their use. Note that CA instruction read and write operations are always generated during the Operand Fetch cycle.

In the following example, instructions I1 to I3 are used to perform a test and set operation. Instructions I4 and I5 are subsequent instructions that cause the processor to branch.

```
I1    dolock 1,0          /* Assert LOCKN and disable interrupts and DMAs for the
                             following two instructions Operand Fetch cycles */
I2    r1=*r2              /* Loading r1 will set the n and z flags for the operand pointed
                             to by r2 */
I3    *r2=r3              /* r3 contains the flag value  to be written back */
I4    if(eq) pcgoto czero /* Flags corresponding to the value in r1 can be tested here */
```

I5 nop

If both instruction and operand accesses were to external memory, the activity and the extent of the assertion of the LOCKN pin for the above sequence of instructions would be:

Access	LOCKN	Comment
I1	1	Fetch dolock 1,0
I2	1	Fetch r1=*r2
I3	1	Fetch *r2=r3
OI2	0	Read operand for I2, *r2
I4	0	Fetch I4
OI3	0	Write operand for I3, *r2
I5	1	Fetch I5

6.6.3 Quad-word Block Move Indicators

A block move is a programmed transfer of a block of data from one location in memory to another location in memory. It is used to move blocks of data from external memory to the DSP3210 internal memory or from the DSP3210 internal memory to external memory. The DSP3210 provides two methods of signaling a quad-word block move operation. When Motorola signaling is selected (pcw[9] = 1), a quad-word transfer is signaled the same as a 68040 burst-inhibited line read. The first word of a group of four words to be transferred sets the MS3—MS2 pins to 11. The remaining three words to be transferred set the MS3—MS2 pins to 00. Secondly, the BLMN pin will be asserted for the first word of the group of four words when selected. The PBD/BLMN pin is selected to be BLMN by setting pcw[15—14] = 00. It is the responsibility of the external bus arbiter to grant the bus to the DSP3210 for the duration of the quad-word transfer.

The instruction sequence used to perform block moves is a **doblock** instruction followed by the DA instruction, **a0=(Z=Y)*a0**, in which the Y operand is written back to memory performing one memory-to-memory transfer per instruction cycle. The doblock loop is decomposed into atomic four word transfers that begin with an address with A3—A2 = 00 and sequence 01, 10, 11. When the do-loop count is four or more and the address has A3—A2 = 00, the start of an atomic quad-word transfer is detected. Interrupts and DMAs are disabled for the next three instruction cycles so that the quad-word transfer is not disrupted. Interrupts and DMA cycles are arbitrated between quad-word transfers. The address of the first read or write location does not have to be quadword aligned, nor does the number of iterations need to be a multiple of 4. Read or writes prior to the detection of a quad-word transfer are standard long word transactions. Reads or writes at the end of the loop when fewer than four words remain are also standard long word transactions.

Due to the DAU pipeline, when performing block moves **to** external memory, at least the last three words and up to as many as the last six words will not be able to be made into quad-word transfers. This is because the remaining block count is compared to the address during the Operand Fetch cycle of the instruction pipeline (See Figure 7-1A Pipeline for a Single DA Instruction). Also, while the last three words are being written subsequent instructions can be fetched invalidating the sequential addressing nature of a quad-word transfer. A block move to external memory should always be followed by three instructions that do not have Z-writes before a subsequent block move is initiated.

The following doblock loop performs a block move of r3 + 1 words:

```
doblock r3                  /* Repeat the following instruction r3 + 1 times */
a0= (*r2++=*r1++)*a0        /* Note that this instruction is cached on-chip; the r1 pointer is to the
                               source, and the r2 pointer is to the destination */
3*nop                       /* Three instructions must be executed between block moves to
                               external memory in order to clear the pipe */
```

Figure 6-10 shows the timing for the bus interface signals for a quad-word transfer from external memory to internal memory. It consists of four consecutive read cycles. Even though the DSP3210 has a bus bandwidth of two words per instructions cycle, when using this type of instruction, it can accept data at one word per instruction cycle . Therefore, two idle states are observed between each bus transaction.

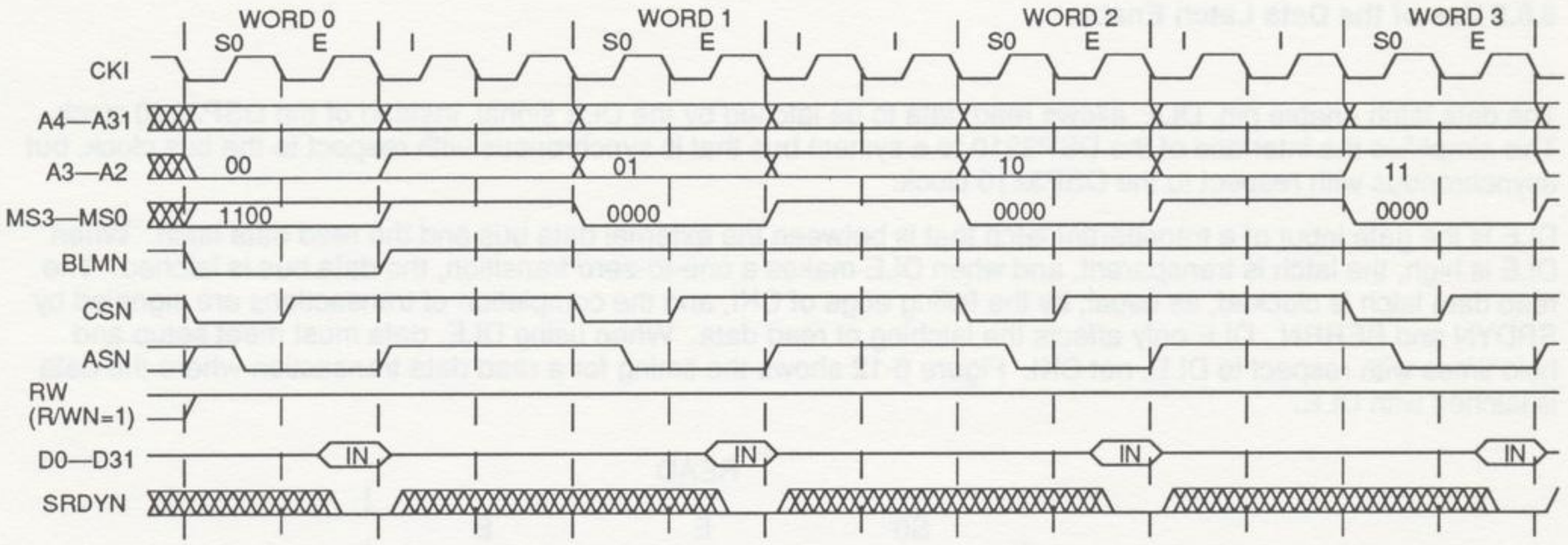


Figure 6-10 Quad-word Block Move Timing

6.6.4 Multiplexed Address and Data Bus

To use the DSP3210 in a multiplexed address and data bus system, AEN and DEN must be used. Basically, an external state machine should control when the address is enabled/disabled for read and write operations, and control when the data is disabled/enabled for write operations. Figure 6-11 shows an example of the timing for a multiplexed address and data bus. AEN and DEN operate asynchronously to the DSP3210 bus protocol. See Section 12 Electrical Characteristics for timing specification for AEN and DEN.

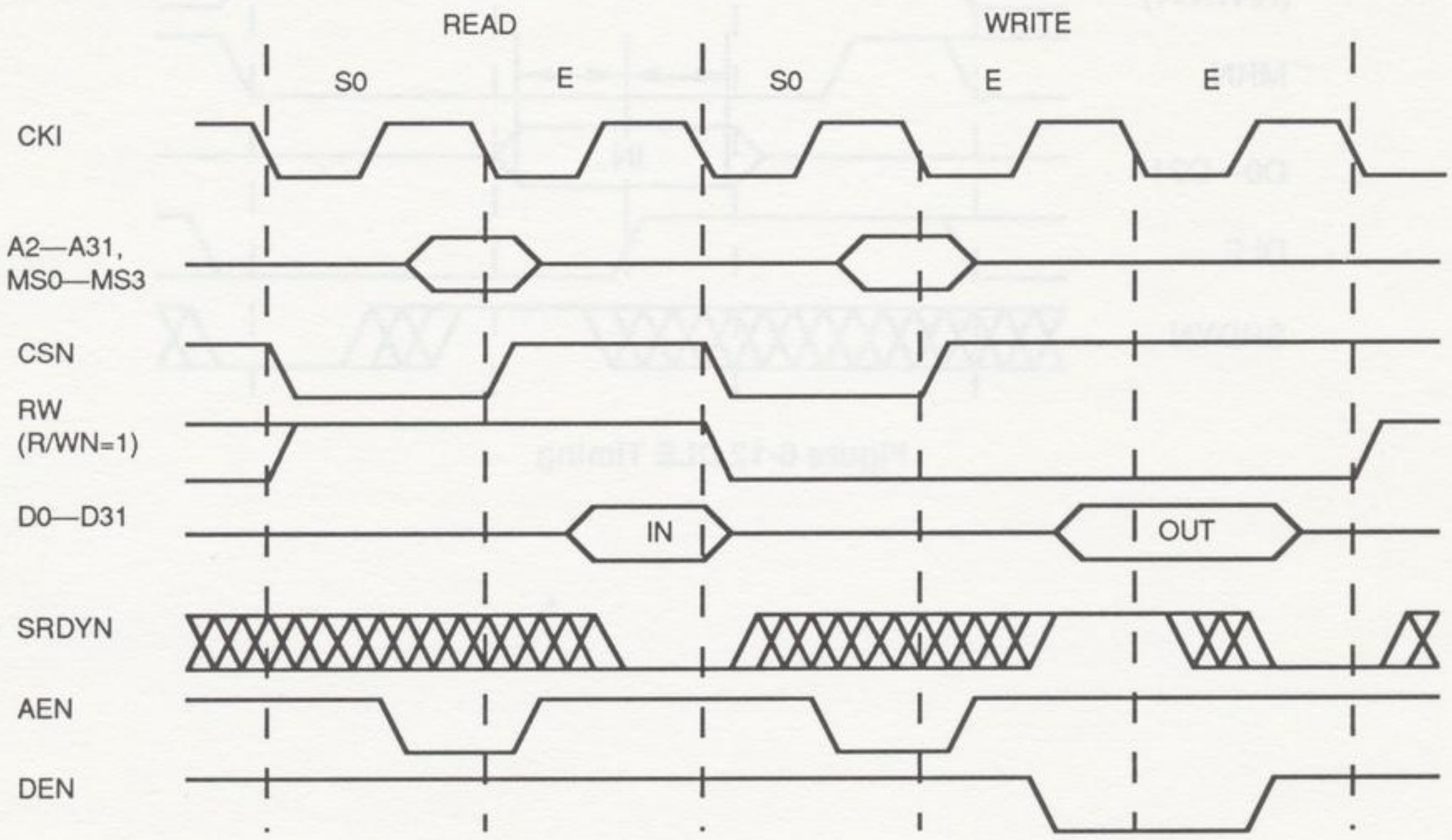


Figure 6-11 Multiplexing the Address Bus and Data Bus

6.6.5 Use of the Data Latch Enable

The data latch enable pin, DLE, allows read data to be latched by the DLE signal, instead of the DSP3210 clock. This simplifies the interface of the DSP3210 to a system bus that is synchronous with respect to the bus clock, but asynchronous with respect to the DSP3210 clock.

DLE is the gate input of a transparent latch that is between the external data bus and the read data latch. When DLE is high, the latch is transparent, and when DLE makes a one-to-zero transition, the data bus is latched. The read data latch is clocked, as usual, by the falling edge of CKI, and the completion of transactions are signaled by SRDYN and BERRN. DLE only affects the latching of read data. When using DLE, data must meet setup and hold times with respect to DLE, not CKI. Figure 6-12 shows the timing for a read data transaction where the data is latched with DLE.

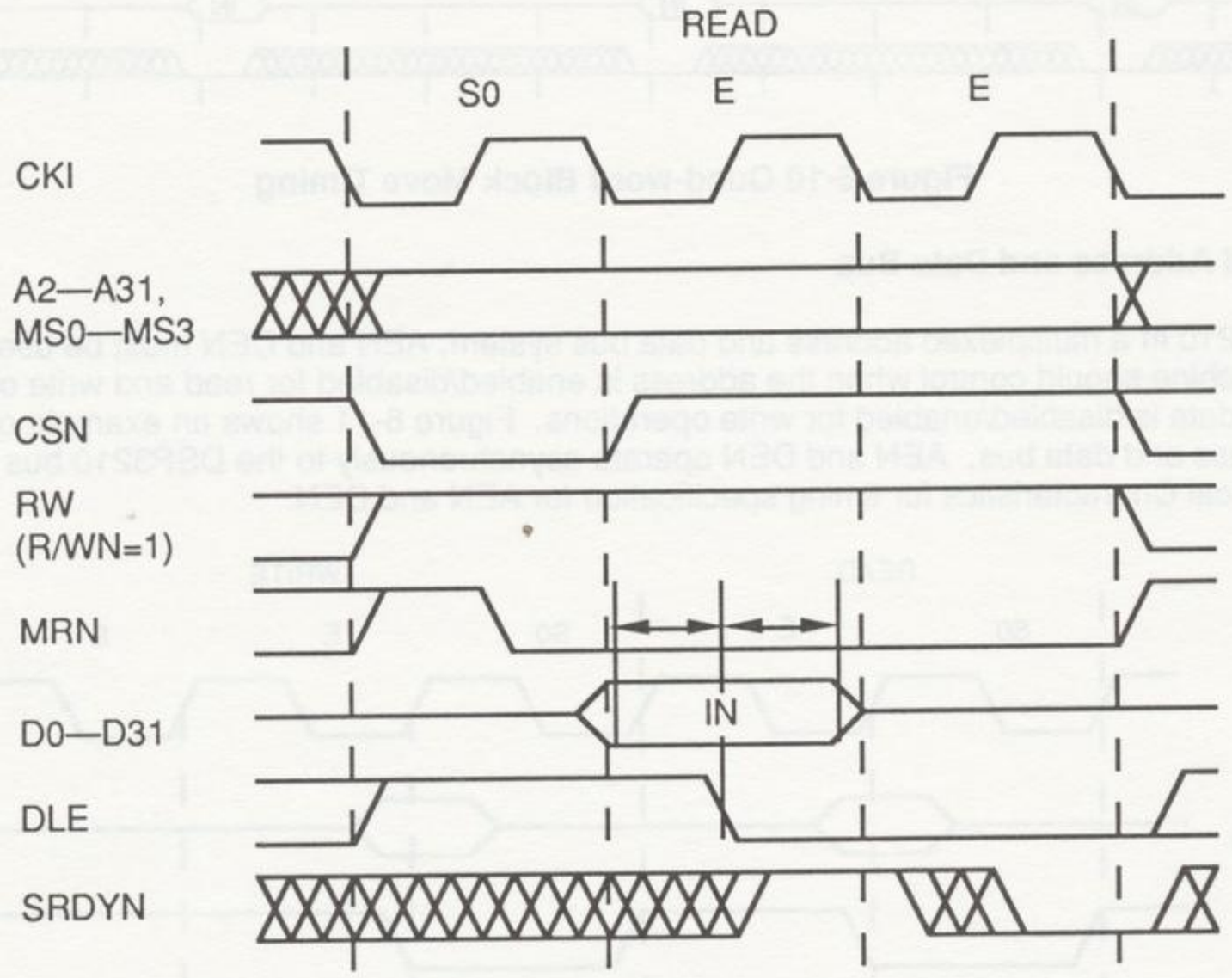


Figure 6-12 DLE Timing

Chapter 7

Processor States

CHAPTER 7. PROCESSOR STATES

7. PROCESSOR STATES 7-1
7.1 Normal Instruction Cycle 7-1
Figure 7-1A Pipeline for a Single DA Instruction 7-1
Figure 7-1B Full Pipeline for Multiple DA Instructions 7-1
7.2 Wait-states 7-1
Figure 7-2 Instruction Cycle with Conflict and External Wait-states 7-2
7.3 Direct Memory Access 7-2
Figure 7-3 DMA Cycle Timing 7-3
7.4 Wait-for-interrupt mode 7-3
7.5 Instruction Sequencing Exceptions 7-3
Figure 7-4 Processing Levels 7-4
Table 7-1 Priority of Exception Processing 7-5
Figure 7-5 Exception Vector Table 7-5
Table 7-2 emr Register Encoding 7-6
7.5.1 Interrupt Exceptions 7-6
Figure 7-6 Interrupt Exception Timing 7-7
7.5.1.1 Quick Interrupt 7-8
7.5.2 Error Exceptions 7-8
7.5.2.1 Double Error Exceptions 7-9
Table 7-3 Registers Affected by Error Exceptions Other Than Reset 7-9
Figure 7-7 Example of Error Exception Process 7-9
7.5.3 Description of Error Exception Sources 7-10
7.5.3.1 Bus Error 7-10
7.5.3.2 Illegal Opcode 7-10
7.5.3.3 Address Error 7-10
7.5.3.4 DAU Underflow/Overflow 7-10
7.5.3.5 NAN 7-11
7.5.4 Reset 7-11
Table 7-4 Registers Affected by Reset 7-11
7.5.4.1 Reset Timing 7-12
Figure 7-8A Power-On Initiates the Reset Sequence 7-12
Figure 7-8B RESTN Initiates the Reset Sequence 7-13

7. PROCESSOR STATES

This chapter describes the modes of operation of the DSP3210 . The DSP3210 is equipped with advanced control features that improve performance and simplify system design. This chapter describes the normal instruction cycle, wait-states, direct memory access (DMA), exceptions (including reset, errors, and interrupts), and wait-for-interrupt (powerdown).

7.1 Normal Instruction Cycle

The DSP3210 has an instruction cycle rate of 1/4 the incoming clock frequency (CKI). Each instruction cycle is divided into four machine states, numbered 0-3, where one state equals one period of CKI. In a single instruction cycle, the DSP3210 can perform an instruction fetch, two operand fetches, and a write to memory. The DSP3210 can execute basically two types of instructions; CA instructions and DA instructions (see Section 4 Instruction Set). Each CA instruction can have up to two memory accesses: (1) memory read (**I** instruction), and (2) memory read or write (**X**-read, **Y**-write). Each DA multiply-accumulate instruction can have up to four memory accesses: (1) memory read (**I** instruction), (2) memory read (**X** operand), (3) memory read (**Y** operand), and (4) memory write (**Z** operand). Because the DAU is pipelined, the four accesses do not all occur in the same instruction cycle for a given instruction. Figure 7-1A shows when address and data are on the **internal** address bus and **internal** data bus for one DA instruction. Figure 7-1B illustrates a full pipe of sequential DA multiply-accumulate instructions. The subscripts refer to instruction numbers; instructions are fetched and executed in ascending order. For example, X_0 is the X operand for instruction number 0, and X_{-1} is the X operand for the instruction previous to instruction number 0.

	Instruction Cycle																							
	Instruction Fetch				Instruction Decode				Operand Fetch				Multiply				Accumulate				Store			
State	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
Address Bus	I						X	Y														Z		
Data Bus				I						X	Y										Z			

Figure 7-1A Pipeline for a Single DA Instruction

	Cycle 0				Cycle 1				Cycle 2				Cycle 3				Cycle 4				Cycle 5	
State	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1
Address Bus	I_0		X_{-1}	Y_{-1}	I_1	Z_{-4}	X_0	Y_0	I_2	Z_{-3}	X_1	Y_1	I_3	Z_{-2}	X_2	Y_2	I_4	Z_{-1}	X_3			Z_0
Data Bus				I_0	Z_{-4}	X_{-1}	Y_{-1}	I_1	Z_{-3}	X_0	Y_0	I_2	Z_{-2}	X_1	Y_1	I_3	Z_{-1}	X_2	Y_2	I_4	Z_0	X_3

Figure 7-1B Full Pipeline for Multiple DA Instructions

$X * Y + aN = aN = Z$

$X_A (Y = Z)$

7.2 Wait-states

The number of states that it takes for an instruction to complete is affected by the introduction of wait-states. Wait-states are introduced when resources required by the instruction are busy and the processor must wait in order to continue processing. There are two types of wait-states: conflict wait-states and external wait-states. Conflict wait-states arise when back-to-back memory requests are made to the same physical memory and the physical memory is either the Boot ROM/MMIO or the external memory. Conflict wait-states **do not** occur when accessing on-chip RAM, RAM0 or RAM1. External wait-states arise when the number of states required to access data in external memory is greater than two or when the DSP3210 is requesting an external memory transaction and it does not have ownership of its bus interface.

The DSP3210 has three physical memories; RAM0/RAM1, External Memory A/B and Boot ROM/MMIO. Some physical memories can introduce conflict wait-states because it takes two states to read or write them and addresses are issued every state. The two physical memories that have this property are the External Memory A/B and the Boot ROM/MMIO. The entire external memory space (Partition A and Partition B) is considered one physical memory, and the combined Boot ROM and Memory-mapped I/O space is considered one physical memory. Wait-states are invisible to the programmer except for their effect of decreasing performance in the DSP3210. However, throughput is degraded only incrementally by the insertion of a conflict wait-state because it is only 1/4 of an instruction cycle.

If external memory is slower than that allowed for zero wait-state operation (2 states), external wait-states are inserted. Also, if the external memory interface is not granted to the processor when an external memory access is required, wait-states are inserted until the bus is granted. Note again that throughput is degraded only incrementally by the insertion of external wait-states because each is only 1/4 of an instruction cycle.

Given the pipelined nature of this device, it is not immediately obvious what successive memory accesses are. Figure 7-1B shows that several DA instructions may be in various stages of execution during a single instruction cycle. To achieve full throughput, no two consecutive accesses can be made to the same physical memory that incurs wait-states. Thus, for example, if the X and Y operands for a given instruction are fetched from the external memory, one conflict wait-state will be inserted between cycle 0-state 3 and cycle 1-state 0. Similarly, if the result for instruction **i** is written to external memory and instruction **i + 4** is fetched from external memory, one conflict wait-state is inserted between cycle 1-state 1, and cycle 1-state 2.

Figure 7-2 shows the effect of conflict and external wait-states on the instruction cycle. In this example, the pipeline is full of DA instructions and all memory accesses are being directed to external memory. Instructions are in logical partition A with zero wait-states, and data is in logical partition B with one wait-state. Conflict wait-states are denoted as Wc, and external wait-states are denoted as WE. Even though there are wait-states, the entire bandwidth of the external memory port is utilized.

	Cycle 0				Cycle 1										
State	0	1	2	3	Wc	WE	0	Wc	WE	1	Wc	2	Wc	WE	3
Address Bus	I_0		X_{-1}	Y_{-1}			I_1			Z_{-4}		X_0		Y_0	
Data Bus				I_0			Z_{-4}			X_{-1}		Y_{-1}		I_1	
External Address Bus		I_0	I_0	X_{-1}	X_{-1}	X_{-1}	Y_{-1}	Y_{-1}	Y_{-1}	I1	I1	Z_{-4}	Z_{-4}	Z_{-4}	
External Data Bus			I_0			X_{-1}			Y_{-1}		I1		Z_{-4}	Z_{-4}	

Figure 7-2 Instruction Cycle with Conflict and External Wait-states

7.3 Direct Memory Access

External devices can access the on-chip memory in the DSP3210, as well as external memory, using direct-memory access (DMA). DMA transfers occur between the memory and the serial I/O port without processor intervention, i.e., using cycle-stealing where the DSP3210 waits for one instruction cycle while the DMA transfer occurs. Two on-chip DMA controllers support memory access via the serial input port and the serial output port.

When DMA is enabled, serial I/O (SIO) transfers occur between the input buffer (ibuf) register and memory and/or between memory and the output buffer (obuf) register without program intervention. The DMA control (dmac) register enables and configures serial I/O DMA activity (see 9.2 DMA Controller). The memory address pointers and the number of transfers to perform for input and output DMA are contained in the input pointer (idp), output pointer (odp), input transfer count (icnt), and output transfer count (ocnt) registers. The serial DMA facility is useful for multichannel applications so that input and output buffers for an entire frame of samples can be established in memory.

Figure 7-3 shows how instruction processing is suspended by the insertion of a DMA cycle. The row labelled IR represents the Instruction Register.

		Cycle 0				DMA Cycle				Cycle 1			
State	3	0	1	2	3	0	1	2	3	0	1	2	3
Address Bus		I_1	Z_{-4}	X_0	Y_0					I_1	Z_{-3}		
Data Bus	I_0	Z_{-4}			I_1		X_0	Y_0		Z_{-3}			I_2
IR		I_0	I_0	I_0	I_0					I_1	I_1	I_1	I_1

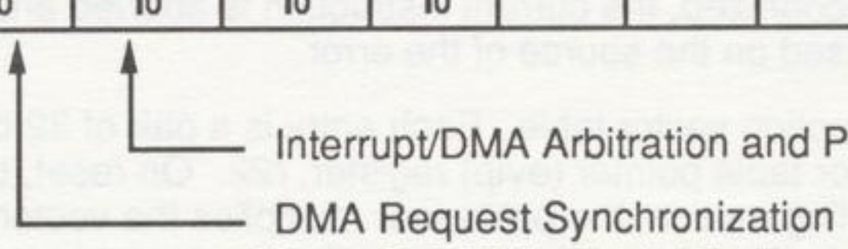


Figure 7-3 DMA Cycle Timing

7.4 Wait-for-Interrupt Mode

The DSP3210 provides a wait-for-interrupt mode so that the power dissipation of the device is reduced when it is not performing data processing. The wait-for-interrupt mode is enabled by the execution of the instruction **waiti**. This instruction will cause the processor to enter continuous DMA-like cycles. If requested, DMA transfers are performed. When any unmasked interrupt exception is detected, the processor exits the wait-for-interrupt mode and executes the instruction following the waiti instruction before branching to the appropriate interrupt routine (interrupt processing may be initiated only after executing an instruction, not after a DMA cycle). Also, the wait-for-interrupt mode is terminated if an error exception is detected.

7.5 Instruction Sequencing Exceptions

The DSP3210 has four distinct instruction processing levels; Base level, Interrupt level, Error level, and Double Error level. Movement between any level is considered as an exception process. Base level instruction processing can be altered by the introduction of interrupt routines. Base level or Interrupt level instruction processing can be altered by the introduction of error handling routines. Error handling can be altered by the introduction of a double error. Figure 7-4 shows the processing levels in the DSP3210.

The DSP3210 handles interrupts using a continuation exception processing model. Single error conditions are handled using an abort exception processing model. Double error conditions are handled by stopping the DSP3210.

The continuation exception model means that the necessary processor context is saved so that when exception processing is completed, the processor can resume at the point where it left off. The abort exception model means that certain processor context is cleared and the current instruction cycle is aborted. The processor can not resume processing where it left off. Register information to locate the cause of the error is maintained so that debugging can be done (refer to Table 7-3). When a double error condition is recognized, the DSP3210 aborts the current instruction, stops, and signals the external system. At this point, the DSP3210 must be reset to start instruction processing.

Interrupt exception, DMA, and instruction processing decisions are accepted at each instruction cycle boundary. In fact, arbitration is done during state 1 in order to decide what to do in the following instruction cycle that starts in state 0. Also, interrupt exception, DMA, or instruction processing decisions are based on the highest priority activity concurrently being requested. Interrupt exceptions are the highest priority, and instruction processing is the lowest. If no interrupt exceptions are requested, then the processor will check to see if a DMA cycle is requested. If no DMA cycle is requested, then a normal instruction cycle is performed. When an interrupt condition is recognized, the DSP3210 saves the processor context in interrupt shadow registers, and the processor branches to the appropriate interrupt handling routine based on the source of the interrupt. When the interrupt routine executes an **ireturn** instruction, normal instruction processing is reinstated by using the context saved in the interrupt shadow registers.

Error exceptions are arbitrated on a state-by-state basis. The detection of an error condition will cause the processor to abort the instruction at its current point of execution; processor level and do-loop control information are reset, and DMA and interrupts are disabled. Table 7-1 shows the priority of exception processing. Exceptions have a fixed priority, but masking can be used to selectively enable desired exceptions. Error and interrupt masking is done in the exception mask register (emr). Instructions that lock out interrupts will not respond to an interrupt for the duration of the lock. Error exceptions can not be locked out, although some exceptions can be masked. When an error condition is recognized, the current instruction is aborted and the processor branches to the appropriate error debug handler based on the source of the error.

The DSP3210 supports a 16-entry exception vector table. Each entry is a pair of 32-bit words starting at the location specified in the exception vector table pointer (evtp) register, r22. On reset, the evtp register is cleared to zero. To branch to the exception handling routine, the processor multiplies the vector number by eight to determine the offset. It adds the offset to the address in the exception vector table pointer (evtp) to obtain the memory address of the exception vector. A branch instruction using this address is executed followed by a nop. Figure 7-5 shows the layout of the exception vector table.

Only the initial reset vector is fixed in the DSP3210's memory map (evtp = address 0 on reset). When modifying the location of the vector table, the new table entries must be established starting at the new base address. Then, the evtp pointer is changed to point to the new base address, and then interrupts are enabled in the emr register. The layout of the emr is shown in Table 7-2.

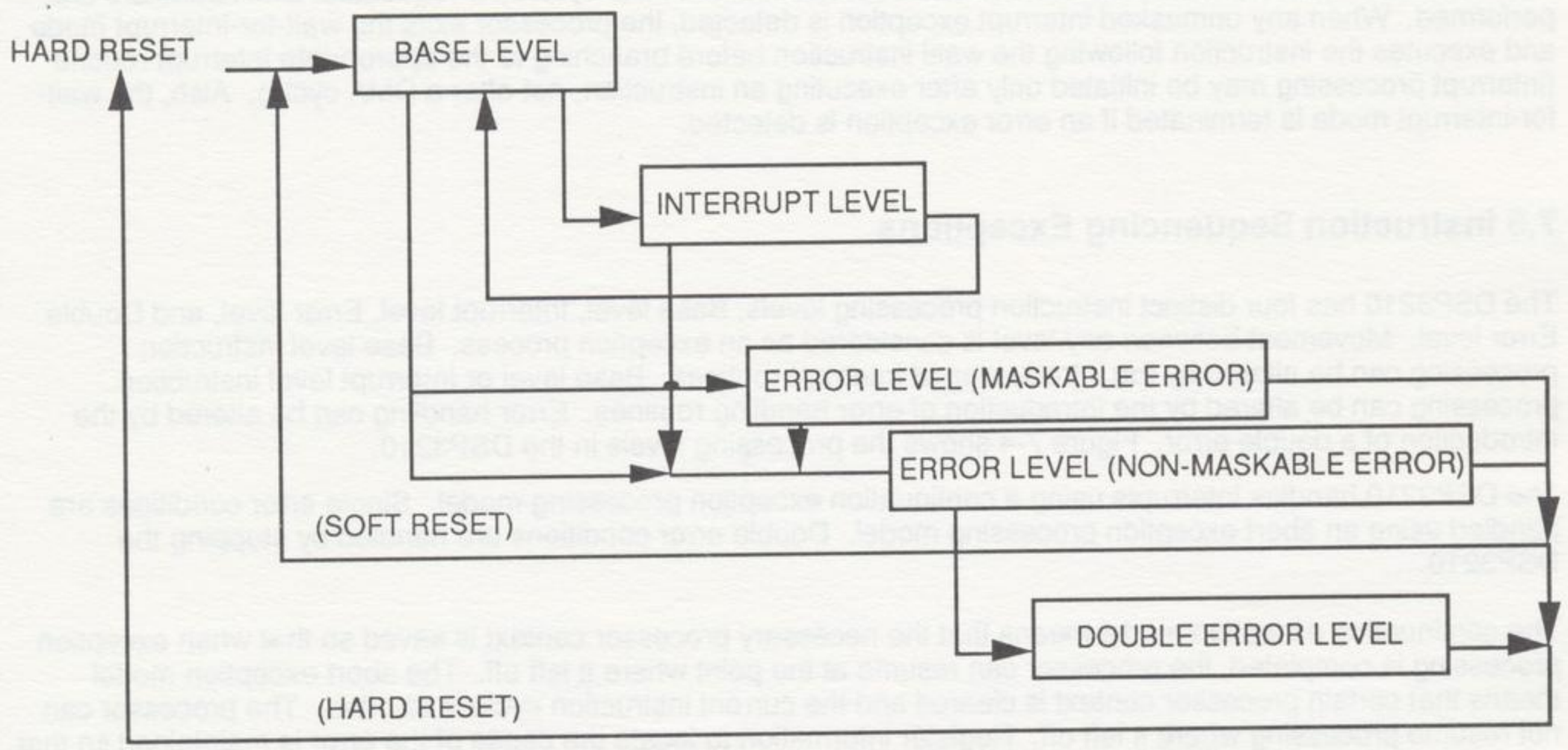


Figure 7-4 Processing Levels

Table 7-1 Priority of Exception Processing

Type	Maskable	Description
Errors		**Abort exception model**
Reset	No	Highest priority error condition
Bus Error	No	
Illegal opcode	No	
Reserved	No	
Address Error	Yes	
DAU Overflow/Underflow	Yes	
NAN	Yes	
Reserved	Yes	Lowest priority error condition
Interrupts		**Continuation exception model**
External Interrupt 0	Yes	Highest priority interrupt condition
Timer	Yes	
Reserved	Yes	
SIO Input Buffer	Yes	
SIO Output Buffer	Yes	
SIO DMA Input Frame	Yes	
SIO DMA Output Frame	Yes	
External Interrupt 1	Yes	Lowest priority interrupt condition

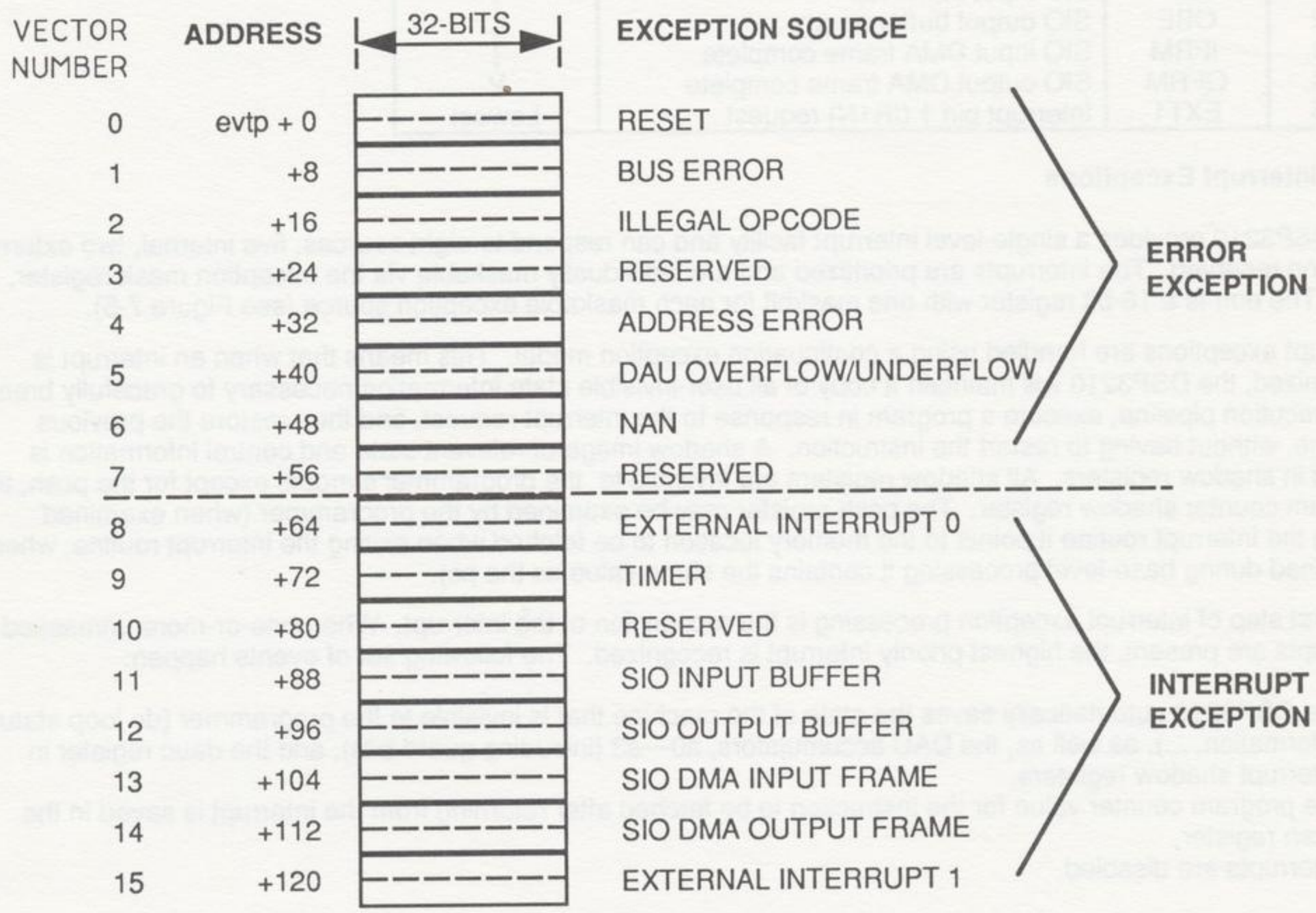


Figure 7-5 Exception Vector Table

Table 7-2 emr Register Encoding

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3 2 1	0
Field	EXT1	OFRM	IFRM	OBE	IBF	RES	TIMER	EXT0	RES	NAN	U/V	AERR	RES	SL

Each bit in this register corresponds to one of the maskable sources for an exception. A value of 1 in a position enables the corresponding interrupt source; a value of 0 disables the source. On reset all bits are set to 0 (zero).

Bit(s)	Field	Function	Priority
0	SL	If 1, select processor level to be visible on IACK[1—0]. Encoding: 00 - Base level 01 - Interrupt level 10 - Error level 11 - Double error level	—
1—3		Reserved	—
4	AERR	Addressing Error	Highest
5	U/V	DAU Underflow/Overflow	|
6	NAN	IEEE NAN	|
7		Reserved	|
8	EXT0	Interrupt pin 0 (IR0N) request	|
9	TIMER	TIMER counted to zero	|
10		Reserved	|
11	IBF	SIO input buffer full	|
12	OBE	SIO output buffer empty	|
13	IFRM	SIO input DMA frame complete	|
14	OFRM	SIO output DMA frame complete	V
15	EXT1	Interrupt pin 1 (IR1N) request	Lowest

7.5.1 Interrupt Exceptions

The DSP3210 provides a single-level interrupt facility and can respond to eight sources, five internal, two external, and one reserved. The interrupts are prioritized and are individually maskable via the exception mask register, emr. The emr is a 16-bit register with one maskbit for each maskable exception source (see Figure 7-5).

Interrupt exceptions are handled using a continuation exception model. This means that when an interrupt is recognized, the DSP3210 will maintain a copy of all user-invisible state information necessary to gracefully break the execution pipeline, execute a program in response to the interrupt request, and then restore the previous pipeline, without having to restart the instruction. A shadow image of relevant state and control information is stored in shadow registers. All shadow registers are invisible to the programmer's model except for the pcsh, the program counter shadow register. The pcsh register may be examined by the programmer (when examined during the interrupt routine it points to the memory location to be fetched when exiting the interrupt routine; when examined during base-level processing it contains the same value as the pc).

The first step of interrupt exception processing is the recognition of the interrupt. When one-or-more unmasked interrupts are present, the highest priority interrupt is recognized. The following set of events happen:

1) the DSP3210 automatically saves the state of the machine that is invisible to the programmer (do loop status information, ...), as well as, the DAU accumulators, a0—a3 (including guard bits), and the dauc register in interrupt shadow registers,
2) the program counter value for the instruction to be fetched after returning from the interrupt is saved in the pcsh register,
3) interrupts are disabled.

The second step of interrupt exception processing is branching to the appropriate interrupt handling routine. To do this the following set of events happen:

1) the exception vector is multiplied by 8 to obtain the table offset,
2) the offset is added to the address in the exception vector table pointer by the instruction **goto evtp + offset** to obtain the memory address of the exception entry and the branch is executed,
3) the goto is followed by a **nop** to account for branch latency.

Internal states that are visible to the programmer (other than the DAU accumulators, a0—a3) and used in the interrupt level must be saved and restored by the interrupt service routine.

The last step of interrupt exception processing is returning to the base processing level. This is done by execution of the ireturn instruction. The ireturn instruction:

1) restores the state of the machine that is not visible to the user, as well as a0—a3 and the dauc register,
2) fetches the instruction pointed to by the pcsh register,
3) causes the next instruction to be executed to come from the instruction shadow register (irsh),
4) re-enables interrupts.

Three instructions are not interruptible:
—CAU register load from memory or I/O register, or register load from memory
e.g. **rD = *rS** or **rD = ior1 or ior1 = *rs**
—Branch instructions, including goto, conditional goto, call and return
e.g. **goto rN**
—Stack pointer increment/decrement,
e.g. **sp = (long) sp++**

Figure 7-6 shows an example of the interrupt exception process. The row labelled IRSH represents the Instruction Shadow Register.

		I0				goto evtp + N				nop				Interrupt Service Routine		ireturn				I1			
State	3	0	1	2	3	0	1	2	3	0	1	2	3	0 3		0	1	2	3	0	1	2	3
Address Bus		I_1		X_0	Y_0		Z_{-3}			I_{I0}										I_2			
Data Bus	I_0					Z_{-3}	X_0	Y_0					I_{I0}										I_2
IR		I_0	I_0	I_0	I_0	goto evtp + p					nop					I_{IN}	I_{IN}	I_{IN}	I_{IN}	I_1	I_1	I_1	I_1
IRSH		I_0	I_0	I_0	I_0	I_1	I_1	I_1	I_1	I_1	I_1	I_1	I_1			I_1	I_1	I_1	I_1	I_1	I_1	I_1	I_1

Interrupt Arbitration and Prioritization

Interrupt Request Synchronization

Figure 7-6 Interrupt Exception Timing

7.5.1.1 Quick Interrupt

Each entry in the exception vector table may contain instructions. In general, these may be used for a goto and the delayed branch latent instruction. Alternatively, by using these two locations (and any entry pair following this that are not being used) for instructions, an interrupt routine with an overhead of three instruction cycles can be performed. An example of a quick interrupt operation performed by the DSP3210 while processing the instruction stream I1, I2, and I3 is shown below:

```
I1                  /* Base level instruction #1 */
I2                  /* Base level  instruction #2
goto evtp + offset  /* overhead branching to interrupt
nop                 /* latent branch instruction
IR1                 /* useful interrupt routine instruction
ireturn             /* no latent instruction executed after ireturn */
I3                  /* Base level  instruction #3 */
```

7.5.2 Error Exceptions

Error exceptions are handled using an abort exception model. The errors that can cause an error exception are a reset (power-on or RESTN asserted), bus error, illegal opcode, address error, DAU overflow or underflow, or the detection of a NAN on an IEEE to DSP32 format conversion. Error exception processing can be entered from the base level or from the interrupt level.

The first step of error exception processing is the recognition of the error. When one-or-more non-maskable errors are present or maskable errors are present and unmasked, the highest priority error is recognized. The following set of events happen:

1) the instruction currently executing is aborted
   a) do-loops are terminated if active,
   b) pending memory transactions are cancelled,
2) on-going memory transactions are completed (but the data is ignored),
3) DMA operations are disabled (wait-for-interrupt is accordingly terminated), and
4) interrupts are disabled.

The second step of error exception processing is to branch to the appropriate error handling routine. To branch to the exception handling routine:

1) the processor multiplies the exception vector number by eight to determine the offset,
2) the offset is added to the address in the exception vector table pointer (evtp) by the instruction **call evtp + offset (r20)** to obtain the memory address of the exception entry and the call is executed,
3) the pc is placed in r20, the error trace register, as a result of the call (the pc represents the address of the prefetched instruction),
4) the call instruction is followed by a **nop** to account for branch latency,

Figure 7-7 shows an example of the error exception process. In this example, an error is detected during state 1 of Io. Io is aborted and the processor state is forced to state 3 for three clock cycles. This is referred to as the error state. Following the error state, the processor executes the call and nop instructions that vector it to the error handling routine. Table 7-3 shows the registers affected by error exceptions other than reset. Note that on a reset error exception, additional registers are cleared as described in 7.5.4 Reset.

The cause of the error exception is broken into two types; non-maskable errors, and maskable errors. When an error exception is taken, the type of error is recorded so that if subsequent errors occur, the processor can respond to them. If the processor is in the error level caused by a maskable error and another maskable error is detected, the error is ignored. If the processor is in the error level caused by a maskable error and a non-maskable error is detected, an error exception is taken to the non-maskable error source handler routine. If the processor is in the error level caused by a non-maskable error and another non-maskable error occurs, the processor executes a double error exception (see 7.5.3 Double Error Exceptions).

The error level processing state can be exited by a software reset of the processor. A software reset of the processor consists of 1) configuring control and data registers to their reset state or any other desired state, 2) resetting the error level to the base level via the **sftrst** instruction, and 3) branching to 0 or any other desired location. The sole function of the sftrst instruction is to change the processor level to the base level. Steps 1 and 3 are performed in software.

7.5.2.1 Double Error Exceptions

If during error exception processing of a maskable error or non-maskable error another maskable error is detected, the processor will ignore it. If during error exception processing of a maskable error a non-maskable error is detected, the processor will initiate another error exception. If during error exception processing of a non-maskable error another non-maskable error is detected, the processor will enter a double error exception. A double error exception will cause the processor to enter a state that can only be left by asserting the RESTN pin or powering down and powering up the DSP3210. The double error exception is signalled by the simultaneous assertion of IACK0 and IACK1.

Table 7-3 Registers Affected by Error Exceptions Other Than Reset

Register	Value	Comments
r20	Previous PC	Result of call evtp + VN*8 (r20).*
pc	evtp + EN*8	Result of call evtp + VN*8 (r20).
tcon[0]	0	Timer decrementing is disabled.
dmac[0,4]	0	SIO DMA is disabled.
pcw[13]	0	Enable writing of pcw register.

* VN is the exception vector number.

		I0			Error State			call evtp + N (20)				nop						
State	3	0	1	2	3	3	3	0	1	2	3	0	1	2	3	0	1	2
Address Bus		I_1										I_{e0}				I_{e1}		
Data Bus	I_0				I_1										I_{e0}			
IR		I_0	I_0					call evtp + N (r20)				nop				I_{e0}	I_{e0}	

Error Exception Detected

Figure 7-7 Example of Error Exception Process

7.5.3 Description of Error Exception Sources

The next sections describe each error exception in detail. Reset is a special case of the error exception and is discussed in Section 7.5.4 Reset.

7.5.3.1 Bus Error

A bus error occurs when an external memory transaction is terminated with BERRN asserted and SRDYN deasserted. When this code is detected, the current bus cycle is terminated and outstanding fetches are invalidated (see 6.4 Bus Error).

7.5.3.2 Illegal Opcode

The opcode of the instruction is defined as the upper 6-bits of the instruction (31—26). Seven of the 64 possible opcodes are defined to be illegal.

	Bits 31—26 Binary	Bits 31—26 Hex
1	0b000000	0x00
2	0b000001	0x01
3	0b000010	0x02
4	0b001111	0x0F
5	0b010110	0x16
6	0b010111	0x17
7	0b100010	0x22

7.5.3.3 Address Error

An address error occurs when a 32-bit word is addressed with an address not divisible by 4, or a 16-bit half-word is addressed with an address not divisible by 2.

7.5.3.4 DAU Underflow/Overflow

Overflow occurs if a DAU operation results in a number that exceeds the range of positive or negative numbers that are representable in the 40-bit format. An overflow occurs if the result is more positive than $2 - 2^{-31} * 2^{127}$ or more negative than $-2 * 2^{127}$. Underflow occurs if a DAU operation results in a number that is smaller than the range of positive or negative numbers that are representable in the 40-bit format. An underflow occurs if the non-zero result is more negative than $1 * 2^{-127}$ and more positive than $-[1 + 2^{-31}]*2^{-127}$.

7.5.3.5 NAN

A NAN is detected in the IEEE to DSP32 floating-point conversion operation if the IEEE number is a NAN (e = 255 and $f \neq 0$ and s = don't care).

7.5.4 Reset

The DSP3210 provides two reset mechanisms, a power-on reset circuit and an external reset pin, RESTN. The following terminology is needed to define reset operation:

Power-on reset: The power-on reset circuit is active for the first 8 rising clock edges after power has been applied ($V_{DD} \geq 3$ V) to the device and it puts the chip in the reset state. After 8 clock cycles, the logic level of RESTN controls DSP3210 operation. If RESTN is low, the DSP3210 remains in the reset state. If RESTN is high, the DSP3210 performs the reset sequence, and enters the base level processing state.

Reset state: The DSP3210 is in the reset state when the internal reset signal is asserted, initializing the internal states of the chip. In the reset state, external memory transactions, bus requests, wait-for-interrupt, and error or interrupt exception processing are immediately terminated. Internal registers that are affected during reset are shown in Table 7-4. This state is entered when either power-on reset is detected, or RESTN is asserted.

Reset Sequence: The reset sequence is the same as step two of the error exception process (see 7.5.1 Error Exceptions), and it consists of executing two internally generated instructions:

(1) **call evtp + 0 (r20)**
(2) **nop**.

Table 7-4 Registers Affected by Reset

Register	Value	Comments
evtp	0	Exception base address set to 0
r20	Previous PC	Result of call evtp + 0 (r20)
pc	0	Result of call evtp + 0 (r20)
dauc[7—0]	0	Round, etc.
pcw[9—0]	0x38f	Maximum waits, 256 word page, etc.
pcw[10]	bio[7]	Set to logic state applied to bio[7]
pcw[11]	bio[6]	Set to logic state applied to bio[6]
pcw[12]	bio[5]	Set to logic state applied to bio[5]
pcw[13]	bio[4]	Set to logic state applied to bio[4]
pcw[15—14]	0b00	Block Move
emr[15—0]	0	All maskable exceptions masked
tcon[7—0]	0	Timer disabled
timer[31—0]	0xffffffff	Maximum value
ioc[31—0]	0	Passive mode, 32-bit data length, etc.
dmac[7—0]	0	SIO DMA disabled
bioc[7—0]	0	BIO pins configured as inputs
bio[1,3,5,7,9,11,13,15]	0	BIO output register bits

7.5.4.1 Reset Timing

A description of the reset pin follows:

RESTN — **Reset** (Open Drain; Active Low). The RESTN pin is an open-drain signal. When the power-on reset circuit is active, the DSP3210 asserts RESTN by pulling it down. The external system may also pull RESTN pin low at the same time or at other times to cause the DSP3210 to enter the reset state. RESTN is an asynchronous input that is internally synchronized to the falling edge of CKI. On a zero-to-one transition of RESTN, the reset sequence is initiated. The rising edge of RESTN is used to latch the signal values on pins bio[7—4] into pcw[10—13], respectively.

On power-on of the DSP3210, on-chip circuitry puts the device in the reset state for the first 8 clock cycles (the first 8 rising clock edges of CKI). At this time, the logic level of RESTN controls the DSP3210's operation. Figure 7-9a shows the timing for power-on reset. Note that all bus interface signals are placed in the high-impedance state during the reset state and for the first 11 clocks periods of the reset sequence. Outputs and Biputs, other than the BIO and the Bus Interface signals, are placed in the high-impedance state while the power-on circuit is active only. No RESTN pulse is required to reset and start the DSP3210 running when using the power-on capability. The RESTN pin must be pulled to VDD through an external resistor.

An externally controlled reset can be initiated at any time. RESTN is an asynchronous input that is internally synchronized to the falling edge of CKI. Figure 7-9b shows the timing for externally controlled reset operations. The minimum pulse width for RESTN is 4 periods of CKI. All BIO pins are immediately put in the high-impedance state so that signal values may be applied on the rising edge of RESTN. Outputs and Biputs other than the BIO and the Bus Interface signals may be placed in the high-impedance state by asserting ZN.

For both power-on reset and externally controlled reset, the BIO[4—7] inputs must be valid prior to the rising edge of RESTN and held following it in order to guarantee correct operation and to initialize pcw[13—10]. The rise time of RESTN between 0.4 V and 2.4 V must be less than 4 ns. With up to 50 pF of external load, a 1K Ω resistor can be used.

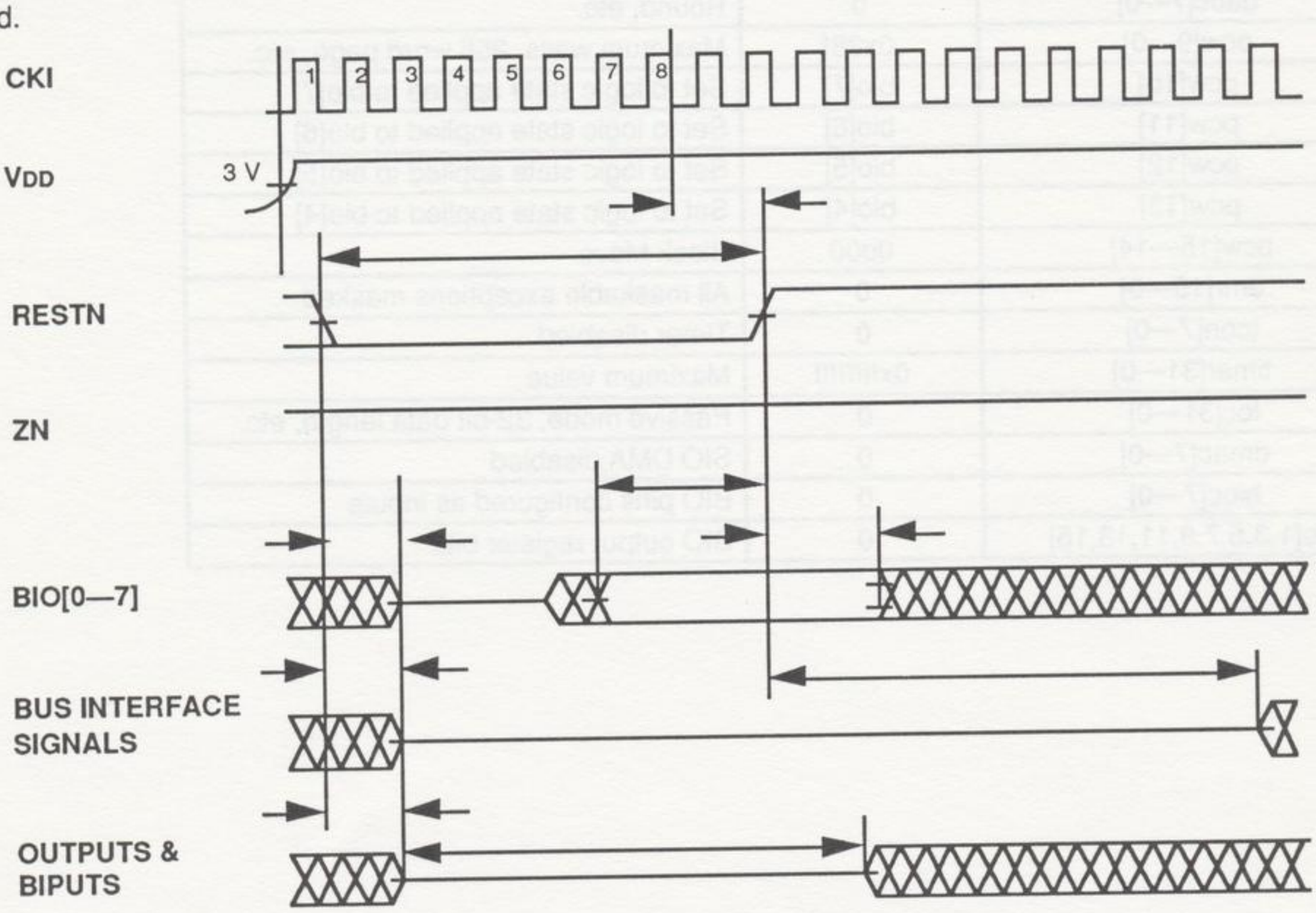


Figure 7-8A Power-On Initiates the Reset Sequence

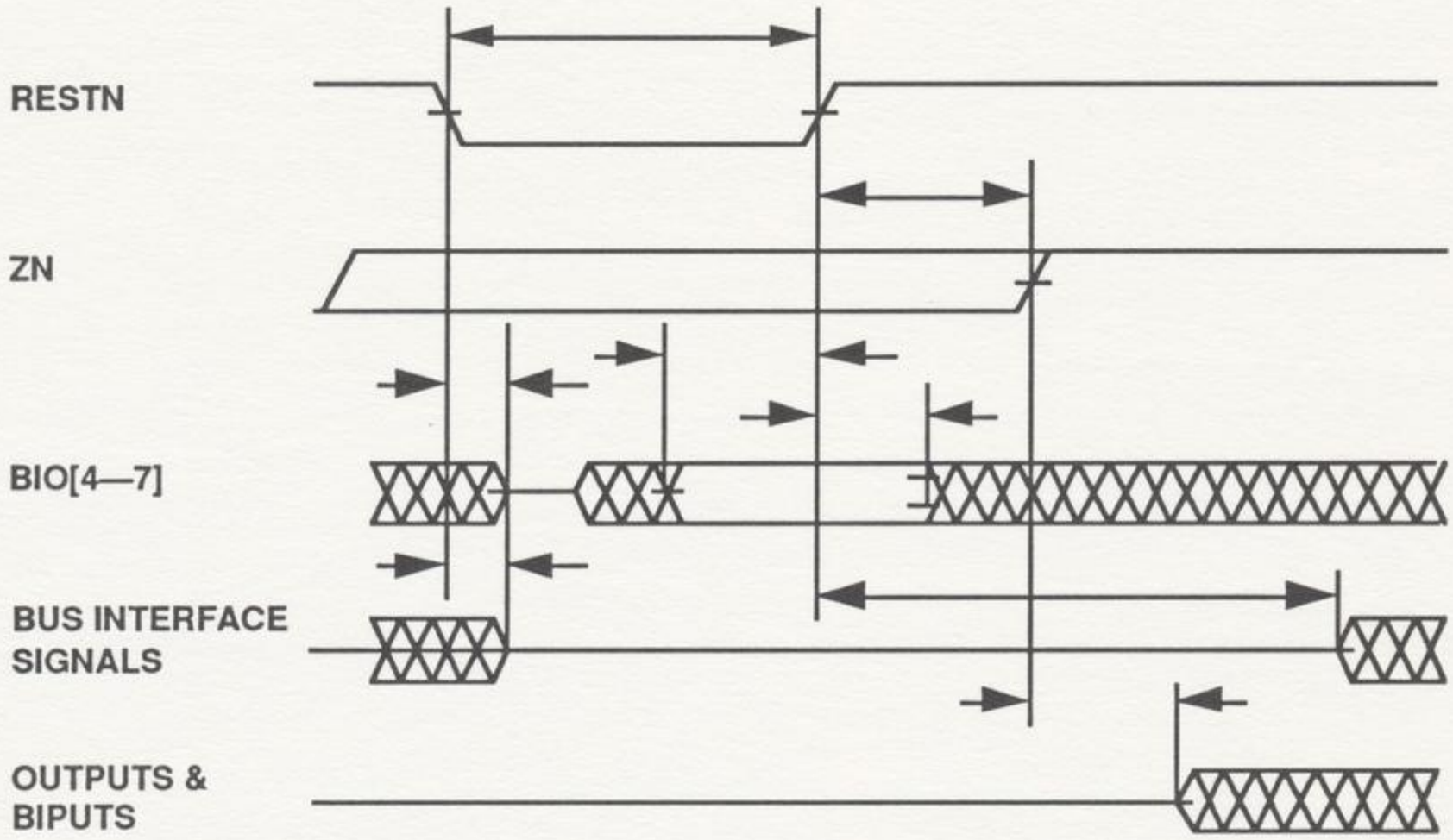


Figure 7-8B RESTN Initiates the Reset Sequence

Chapter 8

Data Processing Functional Units

CHAPTER 8. DATA PROCESSING FUNCTIONAL UNITS

8. DATA PROCESSING FUNCTIONAL UNITS 8-1
8.1 Control Arithmetic Unit (CAU) 8-1
8.1.1 CAU Programming Model 8-1
Figure 8-1 CAU Programming Model 8-1
Table 8-1 ps Register 8-2
Figure 8-2 CAU Register Usage 8-2
8.1.2 CAU Functional Description 8-3
Table 8-2 CAU Operations 8-3
Figure 8-3 Block Diagram of the CAU 8-4
8.1.3 CAU Internal Operation 8-5
Figure 8-4 CAU Internal Pipeline Timing - CA Instruction 8-6
Figure 8-5 CAU Internal Pipeline Timing - DA Instruction 8-6
8.2 Data Arithmetic Unit (DAU) 8-7
8.2.1 DAU Programming Model 8-7
Figure 8-6 DAU Programming Model 8-7
Table 8-3 dauc Register 8-8
Table 8-4 ctr Register 8-8
8.2.2 DAU Functional Description 8-8
Figure 8-7 Block Diagram of the DAU 8-9
Figure 8-8 DAU Floating-Point Formats 8-10
8.2.3 Floating-Point Arithmetic Precision 8-10
Figure 8-9 Precision of DAU Operations (mantissas only) 8-11
8.2.4 Type Conversions 8-12
8.2.4.1 IEEE Standard Floating-Point 8-12
Figure 8-10 IEEE Standard 32-bit Floating-Point Format 8-12
8.2.4.2 Companded 8-13
8.2.4.3 8-bit Unsigned Integer 8-13
8.2.4.4 16-bit Integer 8-13
8.2.4.5 32-bit Integer 8-14
8.2.4.6 Reciprocal Seed 8-14
8.2.5 Rounding Modes 8-14
Figure 8-11 Floating-Point to Integer Mapping for All Rounding Modes 8-15
8.2.6 DAU Internal Operation 8-15
Figure 8-12 DAU Internal Pipeline Timing 8-17

8. DATA PROCESSING FUNCTIONAL UNITS

The DSP3210 provides two data processing units; Control Arithmetic Unit (CAU) and Data Arithmetic Unit (DAU). The CAU performs integer arithmetic, and the DAU performs floating-point arithmetic and data type conversions. Detailed descriptions of these data processing units are contained in this section.

8.1 Control Arithmetic Unit (CAU)

The CAU is responsible for performing address calculations, branching control, and 16- or 32-bit integer arithmetic and logic operations. It is a RISC-like processor consisting of a 32-bit arithmetic logic unit (ALU), 32-bit Barrel Shifter, and a 32-bit wide register stack.

8.1.1 CAU Programming Model

The CAU is used in the execution of both classes of instructions: CA (Control Arithmetic) and DA (Data Arithmetic). CA instructions perform load/store, branching control, and 16- and 32-bit integer arithmetic and logical operations. CA instructions are performed solely by the CAU. DA instructions use the DAU to perform floating-point multiply/accumulate and special function operations and the CAU to generate addresses for the operands. Each DA instructions can have up to three operand memory accesses per instruction, and the CAU is responsible for generating the addresses for the operands. For a detailed description of the instruction set, see Chapter 4 Instruction Set.

Figure 8-1 CAU Programming Model

Table 8-1 ps Register

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	0	0	ir1	ir0	fb	sy	obe	ibf	V	U	Z	N	c	v	z	n

Bit(s)	Field	Function
0	n	CAU **n** flag; means that the ALU result is negative.
1	z	CAU **z** flag; means that the ALU result is zero.
2	v	CAU **v** flag; means that the ALU result overflowed.
3	c	CAU **c** flag; means that the ALU result had a carry out of the MSB.
4	N	DAU **N** flag; means that the DAU result is negative.
5	Z	DAU **Z** flag; means that the DAU result is zero.
6	U	DAU **U** flag; means that the DAU result underflowed.
7	V	DAU **V** flag; means that the DAU result overflowed.
8	ibf	Serial input buffer full flag; means that the serial input buffer is full.
9	obe	Serial output buffer empty flag; means that the serial output buffer is empty.
10	sy	Serial sync set flag; means that the SY pin is high (logic one).
11	fb	Serial frame boundary flag; means that the SIO is at a frame boundary.
12	ir0	Pin IR0N is high (logic one).
13	ir1	Pin IR1N is high (logic one).
15—14	-	Reserved - read as zero.

The CAU register programming model consists of 24 CAU registers and one IO register (see Figure 8-1). The CAU register set consists of r0 (hard-wired to zero), r1—r22 (32-bit registers), pc (program counter)/pcsh (program counter shadow). The IO register is the ps (processor status). The ps register is a 16-bit register and contains CAU flags, DAU flags, and IO flags. The CAU flags are read/writable. The DAU and IO flags are read-only. The position, encoding, and functional definition of the bits in the ps register are available in Table 8-1.

Registers r1—r14 are used as general-purpose registers in CA instructions and as memory pointers (rP) in DA instructions. When used as memory pointers, r1—r14 hold 32-bit addresses. Registers r15—r19 are used as general-purpose registers in CA instructions and as increment registers (rI) in DA instructions. When used as increment registers, r15—r19 hold 32-bit 2's complement values that can post-modify the memory pointers. Registers r0 , r20, r21, and r22 have special properties and can only be used in CA instructions. Register r0 is hard-wired to zero. Register r20 is the error trace register and it is used to hold the program counter address at which an error exception was taken. Register r21 is the stack pointer. Register r22 is the exception vector table pointer. The use of CAU registers in CA and DA instructions is shown in Figure 8-2.

CAU Register	Instruction Type			
	Data Arithmetic		Control Arithmetic	
	Pointer	Increment	General Purpose	Special Purpose
pc				Program counter
r0				Hard-wired zero
r1—r14	X		X	
r15—r19		X	X	
r20			X	Error trace
r21			X	Stack pointer
r22			X	evtp

Figure 8-2 CAU Register Usage

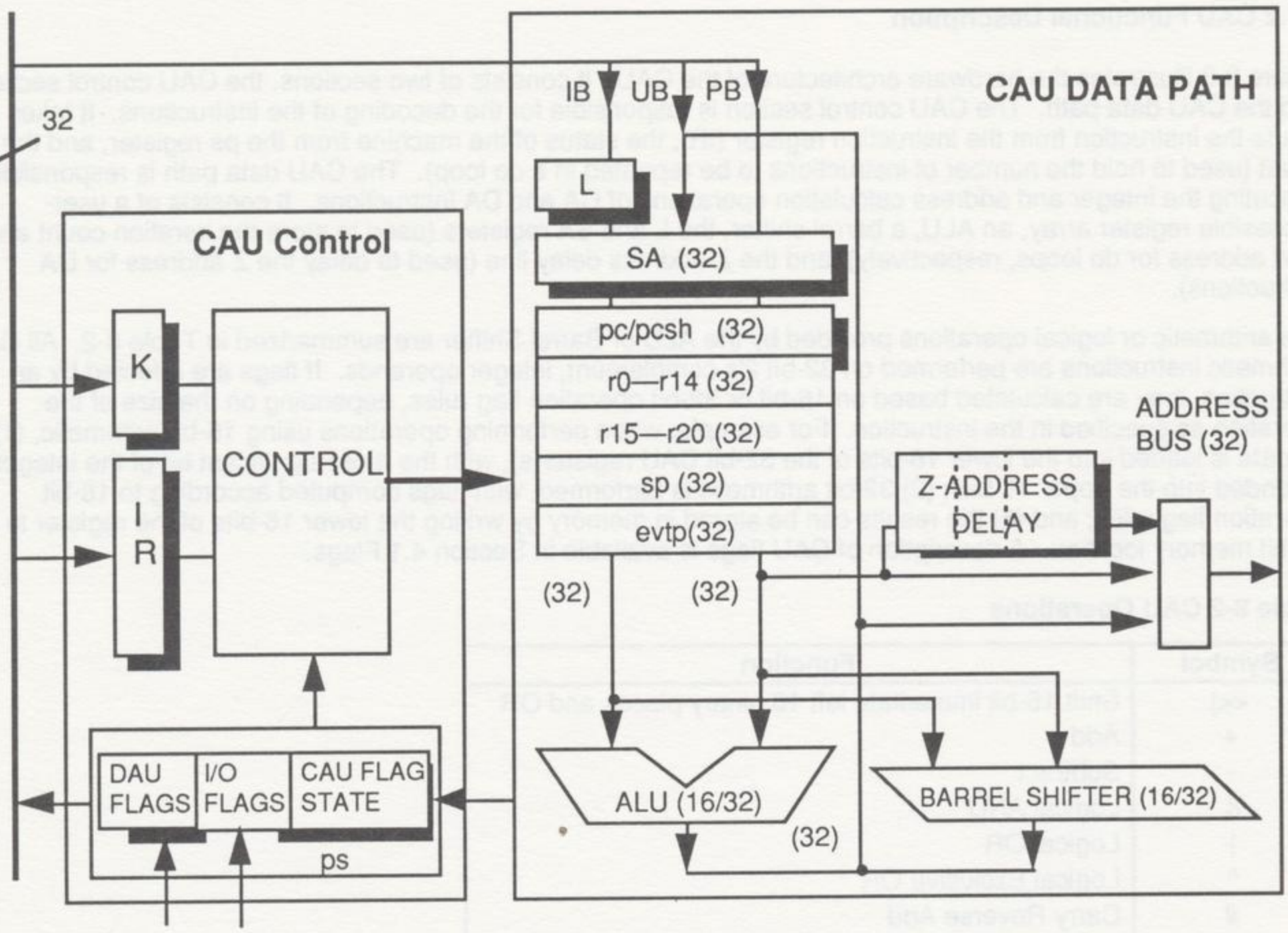
8.1.2 CAU Functional Description

Figure 8-3 illustrates the hardware architecture of the CAU. It consists of two sections, the CAU control section and the CAU data path. The CAU control section is responsible for the decoding of the instructions. It takes as inputs the instruction from the instruction register (IR), the status of the machine from the ps register, and the K count (used to hold the number of instructions to be repeated in a do loop). The CAU data path is responsible for executing the integer and address calculation operations of CA and DA instructions. It consists of a user-accessible register array, an ALU, a barrel shifter, the L and SA registers (used to store the iteration count and start address for do loops, respectively), and the Z-address delay line (used to delay the Z address for DA instructions).

The arithmetic or logical operations provided by the ALU or Barrel Shifter are summarized in Table 8-2. All CA arithmetic instructions are performed on 32-bit 2's complement, integer operands. If flags are affected by an instruction, they are calculated based on 16-bit or 32-bit operation flag rules, depending on the size of the operation as specified in the instruction. For example, when performing operations using 16-bit arithmetic, (1) 16-bit data is loaded into the lower 16-bits of the 32-bit CAU register(s), with the most significant bit of the integer extended into the upper 16 bits; (2) 32-bit arithmetic is performed, with flags computed according to 16-bit operation flag rules; and (3) the results can be stored in memory by writing the lower 16-bits of the register to a 16-bit memory location. A description of CAU flags is available in Section 4.1 Flags.

Table 8-2 CAU Operations

Symbol	Function
<<|	Shift 16-bit immediate left 16 binary places and OR
+	Add
-	Subtract
&	Logical AND
|	Logical OR
^	Logical Exclusive OR
#	Carry Reverse Add
<<	Logical Shift Left
>>	Logical Shift Right
$>>	Arithmetic Shift Right
/2	Arithmetic right shift by 1
*2	Multiply by 2
<<<1	Rotate Left through Carry
>>>1	Rotate Right through Carry
{+,-}1	Increment/Decrement



LEGEND:

IR	Instruction Register	r0	Zero
K	K counter register	r1—r14	CAU general-purpose register, DAU pointer register
L	L counter register		
SA	Start Address	r15—r19	CAU general-purpose register, DAU increment register
IB	Increment Bus		
UB	Update Bus	r20	CAU general-purpose, Error trace
PB	Pointer Bus	sp (r21)	Stack pointer
		evtp(r22)	Exception vector table pointer
		ps	Processor Status

Registers in upper case are not user accessible. ▭ Shadowed during interrupts

Figure 8-3 Block Diagram of the CAU

8.1.3 CAU Internal Operation

The internal operation of the CAU is best understood by describing the operation of its instruction set. The CA instructions consist of three types: arithmetic/logical, data move, and control instructions. Arithmetic/logical instructions are performed as register to register operations. Many of the CA arithmetic instructions can be performed using three different registers: two source registers (rS1, rS2) and one destination register (rD). CA instructions may also be conditionally executed based on flags established in previous instruction cycles. Data move instructions specify 8-, 16-, or 32-bit data transfers between CAU registers and memory, CAU register and I/O registers, and I/O registers and memory. Control instructions alter the program flow by modifying the pc (program counter).

Figure 8-4 illustrates the CAU pipeline for executing two instructions of the form:

rD – rS
if(eq) rD = rS1 + rS2

The first instruction subtracts the contents of the source register (rS) from the destination register (rD), to set/clear the CAU zero (z) flag. The second instruction checks the z flag, and if it is set (i.e., the result of the previous instruction was zero), adds the contents of rS1 and rS2, and stores the result in rD. If the flag is not set, the contents of rD are unaffected by this instruction.

During execution of DA instructions, the CAU generates up to four addresses, one address in each of the four states of an instruction cycle. In each state, the CAU can add the contents of two registers: a pointer selected from r1—r14 (placed on the pointer bus) and an increment selected from r15—r19 or a fixed increment value (placed on the increment bus). The CAU employs a 3-stage pipeline to (1) fetch from a register(s), (2) operate on the fetched operand(s), and (3) store the result in a register. Because of this pipelining, the preceding pointer is updated (result placed on the update bus), while the next pointer and increment are being accessed.

Figure 8-5 shows the CAU pipeline for executing a DA instruction of the form:

$$Z = aN = aM + Y * X$$

Since the result of the multiply-accumulate operation is not available on the data bus until state 0 of instruction cycle I_5 and the destination address is calculated during state 1 of instruction cycle I_2, the address is latched and delayed for three instruction cycles before being placed on the address bus. This three instruction cycle delay is performed by the Z-address delay block shown in Figure 8-3.

Except for the register load from memory and branch operations that have a latency of one instruction cycle, execution of a CA instruction is completed by the CAU before the next CA instruction begins execution. This simplifies use of the CAU for logic and control operations.

		Cycle 0				Cycle 1			
State	3	0	1	2	3	0	1	2	3
Pointer Bus	pc_1		rD		pc_2		rS1		pc_3
Increment Bus	+4		rS		+4		rS2		+4
ALU		+				+			
Update Bus			pc_2				pc_3		?rD
Address Bus		pc_1				pc_2			
Data Bus	I_0				I_1				I_2
IR		I_0	I_0	I_0	I_0	I_1	I_1	I_1	I_1
Flags					z				?

For instructions in the form of:

I_0: rD-rS

I_1: if(eq) rD = rS1 + rS2

where rD = destination register
rS, rS1, rS2 = source register
eq = CAU condition equal to zero (z flag)
? = modification of register and flag is conditional
pc = program counter
+ = additional operation
− = subtraction operation

Figure 8-4 CAU Internal Pipeline Timing - CA Instruction

		Cycle 0				Cycle 1				Cycle 2, 3	Cycle 4			
State	3	0	1	2	3	0	1	2	3		0	1	2	3
Pointer Bus	pc_1		rP_x	rP_y	pc_2	rP_z			pc_3					
Increment Bus	+4		rS	rI_y	+4	rI_z			+4					
ALU		+		+	+	+	+							
Update Bus			pc_2		rP_x	rP_y	pc_3	rP_z						
Address Bus		pc_1		X	Y	pc_2						Z		
Data Bus	I_0				I_1		X	Y	I_2		Z			
IR		I_0	I_0	I_0	I_0	I_1	I_1	I_1	I_1		I_4	I_4	I_4	I_4

For instructions in the form of:

I_0: $*rP_z{+}{+}rI_z$ = aN = aM + $*rP_y{+}{+}rI_y$ * $*rP_x{+}{+}rI_x$

where rP_z, rP_y, rP_x = pointer register for DAU - Z, Y, and X operands
rI_z, rI_y, rI_x = increment register for DAU - Z, Y, X operands
pc= program counter
+= additional operation

Figure 8-5 CAU Internal Pipeline Timing - DA Instruction

8.2 Data Arithmetic Unit (DAU)

The DAU is the primary execution unit for signal processing algorithms. This unit contains a 32-bit floating-point multiplier, a 40-bit floating-point adder, four 40-bit accumulators, Data Type conversion hardware, and two user-accessible registers, dauc (DAU control register) and ctr (Clip-Test register). The multiplier and adder work in parallel to perform 16.7 million computations per second of the form (a = b + c * d). The multiplier and adder each produce one result per instruction cycle. The DAU contains a four stage pipeline: Fetch, Multiply, Add, and Store. Thus, in any instruction cycle, the DAU may be processing four instructions, each in a different stage of execution.

8.2.1 DAU Programming Model

The register programming model for the DAU consists of 4 DAU registers, called accumulators, and two IO registers, the dauc, DAU control register and the ctr, clip-test register. Figure 8-6 shows the register programming model for the DAU. Note that all registers in the DAU are shadowed during interrupts. Each accumulator is a 40-bit register consisting of a 32-bit mantissa and an 8-bit exponent.

The dauc is used to select the 8-bit data type conversion performed for the instructions **ic** and **oc**, the truncation mode to be used when converting floating-point data to integer, and how writes are handled in conditional DA instructions, **aeq**, **alt**, and **agt**. Linear byte conversion, μ-law, and A-law share the same instruction encoding (Z=aN=ic(Y);) and are therefore distinguished only by the dauc setting. Choosing linear byte input and/or output mode overrides the corresponding μ-law/A-law setting. Loading the dauc register from memory affects the following instruction. The meaning for the settings of the dauc register are given in Table 8-3. The dauc register is cleared on reset.

The ctr stores the last six values for the DAU **N** flag. It is used for 2-D and 3-D clip-test algorithms to compute the outcode. This outcode can then be read into a CAU register for further operations. The meaning for each bit of the ctr register is given in Table 8-4.

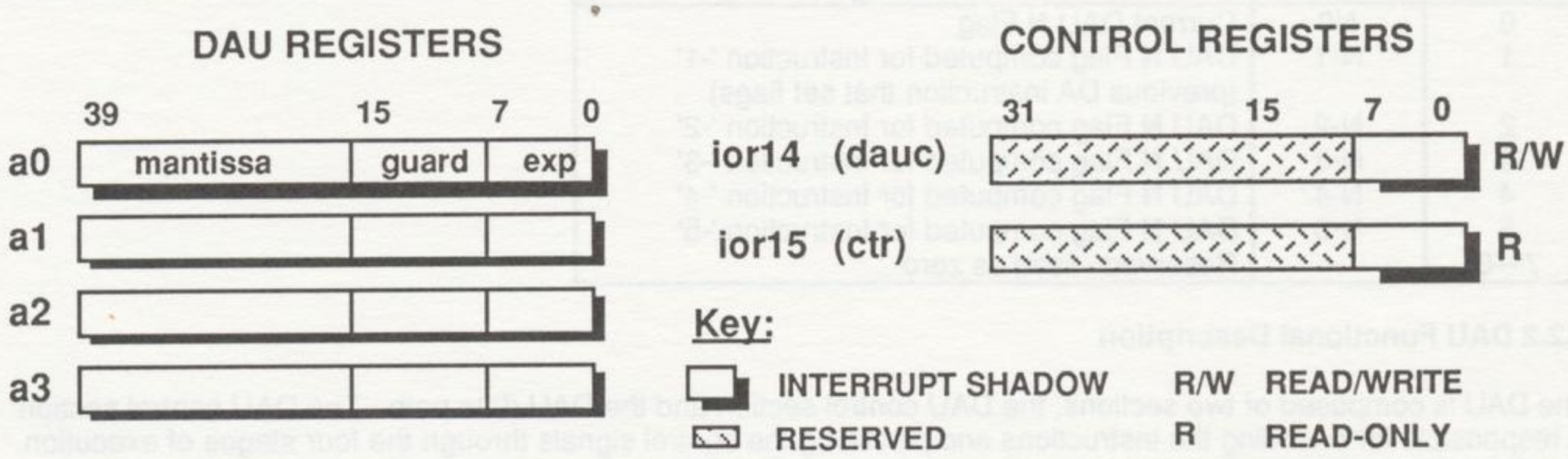


Figure 8-6 DAU Programming Model

Table 8-3 dauc Register

Bit	7	6	5 4	3 2 1 0
Field	0	COND	ROUND	CONV

Bit(s)	Field	Encoding	Function
3—0	CONV	x0x0	µ-law input conversion.
		x0x1	A-law input conversion.
		x1xx	Unsigned linear byte input conversion.
		0x0x	µ-law output conversion.
		0x1x	A-law output conversion.
		1xxx	Unsigned linear byte output conversion.
5—4	ROUND	x0	Round-to-nearest on float-to-integer conversions.
		01	Truncate towards -∞ on float-to-integer conversion.
		11	Truncate towards 0 on float-to-integer conversion.
6	COND	0	DA instructions *ifalt*, *ifaeq*, and *ifagt* have unconditional Z-writes.
		1	DA instructions *ifalt*, *ifaeq*, and *ifagt* have conditional Z-writes.
7	0	0	Must be set to zero; Read as zero.

Table 8-4 ctr Register

Bit	7	6	5	4	3	2	1	0
Field	0	0	N-5	N-4	N-3	N-2	N-1	N0

Bit(s)	Field	Meaning
0	N0	Current DAU **N** Flag
1	N-1	DAU **N** Flag computed for Instruction '-1' (previous DA instruction that set flags)
2	N-2	DAU **N** Flag computed for Instruction '-2'
3	N-3	DAU **N** Flag computed for Instruction '-3'
4	N-4	DAU **N** Flag computed for Instruction '-4'
5	N-5	DAU **N** Flag computed for Instruction '-5'
7—6	-	Reserved - read as zero

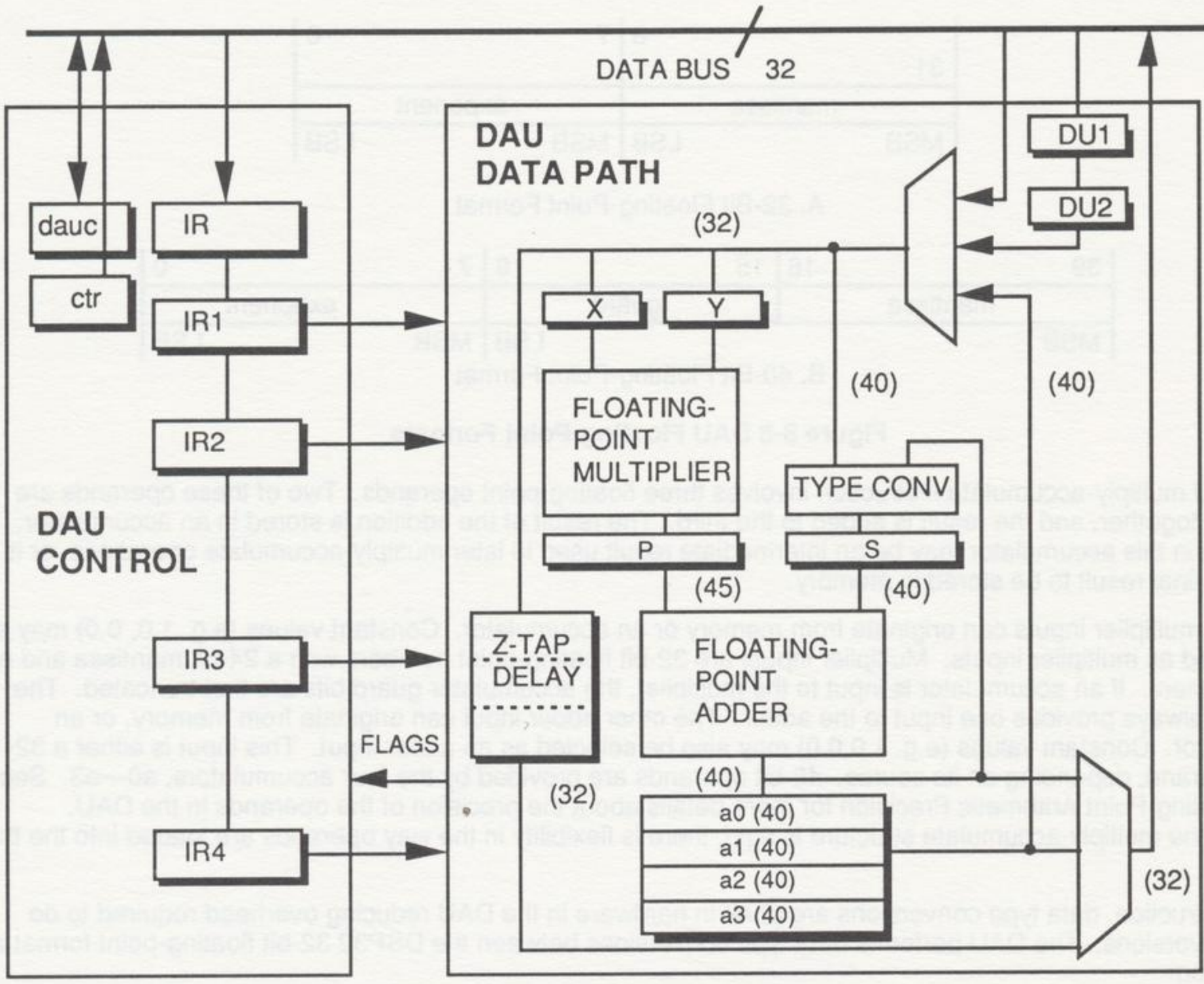
8.2.2 DAU Functional Description

The DAU is composed of two sections, the DAU control section and the DAU data path. The DAU control section is responsible for decoding the instructions and pipelining the control signals through the four stages of execution. The DAU data path is responsible for performing the floating-point arithmetic and data type conversions. Figure 8-7 illustrates the hardware architecture of the DAU.

The DAU supports two floating-point formats, single precision (32-bit) and extended single precision (40-bit). Extended single precision provides 8 additional mantissa guard bits. The four 40-bit accumulators, a0—a3, reduce round-off problems to ensure 24-bit precision. Post-normalization logic transparently shifts binary points and adjusts exponents to prevent inaccurate rounding of bits when the floating-point numbers are added or multiplied, eliminating concerns like scaling and quantization error. All normalization is done automatically, so the result in the accumulator is fully normalized.

The DAU contains both 32-bit and 40-bit registers and buses to support two floating-point formats, 32-bit and 40-bit extended precision floating-point. The 40-bit format provides 8 additional mantissa guard bits used by the DAU in accumulate operations. Figure 8-8 shows the two floating-point formats.

When a 40-bit bus connects to a 32-bit register or multiplexer, the guard bits are excluded (truncated). For example, when writing a 40-bit accumulated result to memory, the result is first truncated to 32-bits. When a 32-bit register drives a 40-bit bus, the guard bits (bits 15—8) are zeroed on the 40-bit bus.



LEGEND:

IR	Instruction Register	X	Multiplier input register
IR1	IR for instr. cycle 1	Y	Multiplier input register
IR2	IR for instr. cycle 2	P	Product register
IR3	IR for instr. cycle 3	S	Special function register
IR4	IR for instr. cycle 4	dauc	DAU Control Register
DU1,DU2	Input operand delay registers	ctr	Clip-test Register
		a0—a3	Accumulators 0—3

Registers in upper case are not user accessible. ▭ Shadowed during interrupts.

Figure 8-7 Block Diagram of the DAU

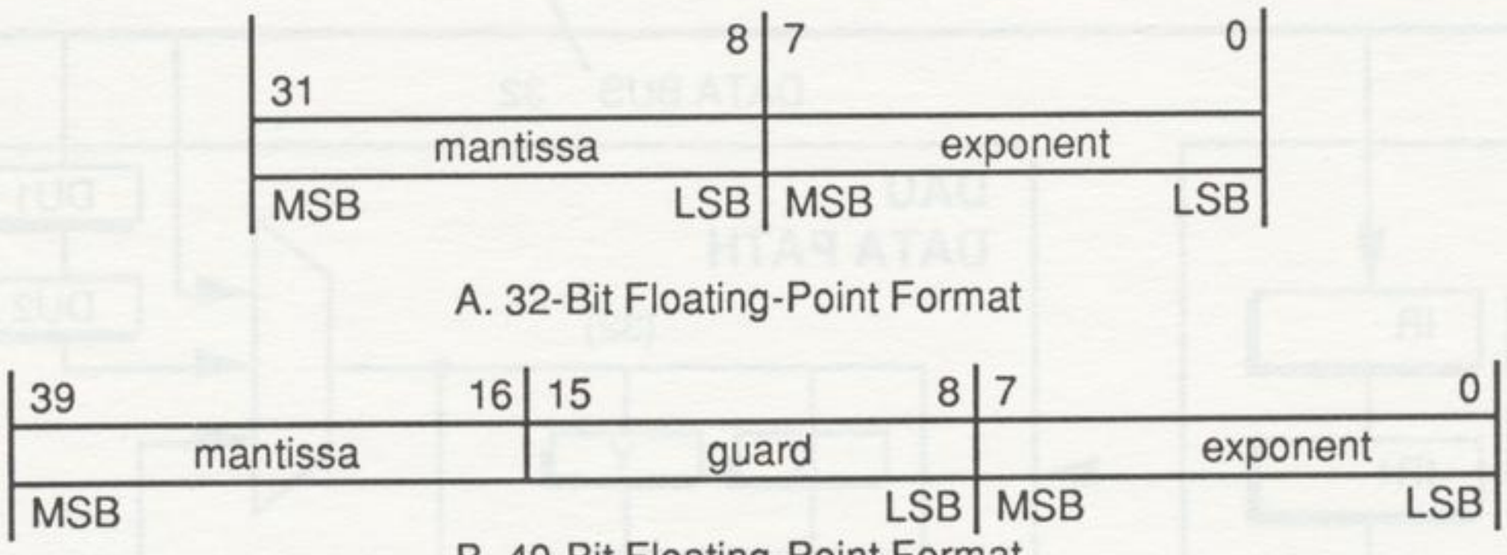


Figure 8-8 DAU Floating-Point Formats

Each DAU multiply-accumulate instruction involves three floating-point operands. Two of these operands are multiplied together, and the result is added to the third. The result of the addition is stored in an accumulator. The value in this accumulator may be an intermediate result used in later multiply-accumulate operations, or it may be a final result to be stored in memory.

The DAU multiplier inputs can originate from memory or an accumulator. Constant values (e.g. 1.0, 0.0) may also be selected as multiplier inputs. Multiplier inputs are 32-bit floating-point numbers with a 24-bit mantissa and an 8-bit exponent. If an accumulator is input to the multiplier, the accumulator guard bits are first truncated. The multiplier always provides one input to the adder. The other adder input can originate from memory, or an accumulator. Constant values (e.g. 1.0,0.0) may also be selected as an adder input. This input is either a 32- or 40-bit operand, depending on its source. 40-bit operands are provided by the four accumulators, a0—a3. See 8.2.3 Floating-Point Arithmetic Precision for more details about the precision of the operands in the DAU. Although the multiply-accumulate structure is rigid, there is flexibility in the way operands are loaded into the three inputs.

Single instruction, data type conversions are done in hardware in the DAU reducing overhead required to do these conversions. The DAU performs data type conversions between the DSP32 32-bit floating-point format and the folowing:

—IEEE P754 standard 32-bit floating-point
—16- and 32-bit integer
—8-bit u-law and A-law formats
—8-bit unsigned integer

The DAU also provides an instruction to convert a 32-bit floating-point operand to a 3-bit seed value used for reciprocal approximation in division operations.

8.2.3 Floating-Point Arithmetic Precision

Each DAU multiply-accumulate instruction (of the form A = S + Y * X) involves three floating-point operands. Two of these operands (X and Y) are multiplied together, and the result (P) is added to the third (S). The result of the addition is stored in one of four accumulators (a0—a3). The contents of this accumulator may be an intermediate result used in later multiply-accumulate operations, or a final result to be stored in memory.

Figure 8-9 illustrates the precision of the floating-point operands and results involved in DAU multiply-accumulate operations. (Because precision is a function of the mantissa length, discussion of the exponent is omitted here.) An implied binary point is included in the figure. Bits to the right of the binary point constitute the fractional part of the number, while those to the left constitute the integer part.

Recall that the magnitude of the mantissa (M), in the multiplier inputs (X and Y), adder input (S) and adder result (a0—a3) is always normalized such that $-2 \le M < -1$ or $1 \le M < 2$. Thus, the leading 1 (positive mantissa) or leading 0 (negative mantissa) does not appear explicitly in the floating-point word, and is called an **implicit** bit.

The result of a multiplication (P) could have a denormalized mantissa (e.g. in cases such as $-2 * -2 = 4$ and $-1 * 1 = -1$), and thus three integer bits are retained in P. Normalization is done automatically in the adder, so the accumulated result is fully normalized. The adder inputs (P and S) and result (a0—a3) contain eight mantissa guard bits in addition to the standard 24 fractional bits in the mantissas.

If the result of a multiply-accumulate operation is written to memory, its guard bits are truncated. If an accumulator is input to the multiplier, its guard bits are also truncated. There is an option to first invoke the round instruction on the accumulator containing the result. The guard bits are useful when the accumulator is input to the adder. Here, the full width accumulator, including guard bits, is used.

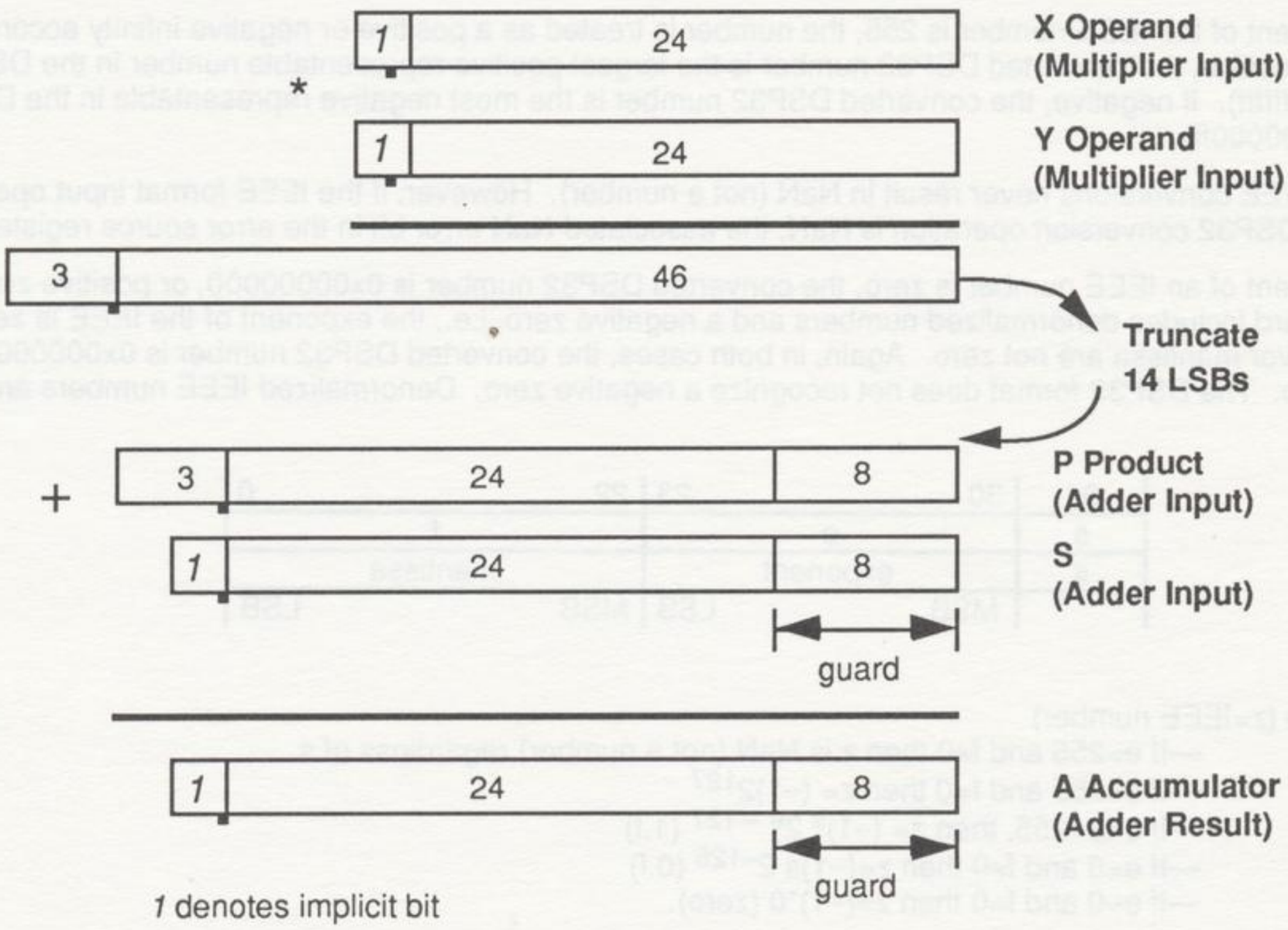


Figure 8-9 Precision of DAU Operations (mantissas only)

8.2.4 Data Type Conversions

Several data type conversions that have traditionally added considerable software overhead are handled by the DSP3210 in hardware. The DAU provides single-cycle instructions that perform data-type conversion between the DSP32 internal 32-bit floating-point format and the IEEE P754 standard 32-bit floating-point format, the 8-bit companded μ- and A-law formats, the 8-bit unsigned linear format, and the 16- and 32-bit integer formats. The following sections describe these operations in more detail.

8.2.4.1 IEEE Standard Floating-Point

Although the DSP3210 has its own internal floating-point format, the DSP3210 is compatible with the IEEE P754 standard 32-bit floating-point format. The device contains on-chip hardware that supports format conversion to and from the IEEE format in a single instruction cycle. The single-precision IEEE standard floating-point format has a sign-magnitude mantissa and an 8-bit biased exponent. Figure 8-10 shows this format.

The floating-point format used by the DSP3210 is referred to as the DSP32 format. It is the format used by the DSP32, DSP32C, and the DSP3210. Conversions between the IEEE standard and DSP32 formats adhere to the following rules:

—If the exponent of the IEEE number is 255, the number is treated as a positive or negative infinity according to its sign. If positive, the converted DSP32 number is the largest positive representable number in the DSP32 format (0x7fffffff). If negative, the converted DSP32 number is the most negative representable in the DSP32 format (0x800000ff).

—DSP32 to IEEE conversions never result in NaN (not a number). However, if the IEEE format input operand in an IEEE to DSP32 conversion operation is NaN, the associated NaN error bit in the error source register is set.

—If the exponent of an IEEE number is zero, the converted DSP32 number is 0x00000000, or positive zero. The IEEE standard includes denormalized numbers and a negative zero, i.e., the exponent of the IEEE is zero, but the sign and/or mantissa are not zero. Again, in both cases, the converted DSP32 number is 0x00000000 or positive zero. The DSP32 format does not recognize a negative zero. Denormalized IEEE numbers are not supported.

31	30 23	22 0
s	e	f
s	exponent	mantissa
	MSB LSB	MSB LSB

where (z=IEEE number)

—If e=255 and f≠0 then z is NaN (not a number) regardless of s.
—If e=255 and f=0 then z= $(-1)2^{127}$
—If 0<e<255, then z= $(-1)^s\ 2^{e-127}\ (1.f)$
—If e=0 and f≠0 then z=$(-1)s\ 2^{-126}\ (0.f)$
—If e=0 and f=0 then z=(−1)*0 (zero).

Figure 8-10 IEEE Standard 32-bit Floating-Point Format

8.2.4.2 Companded

On-chip hardware supports the conversion to and from two 8-bit compressed PCM data formats: μ-law and A-law.

μ-law

The format of an 8-bit, μ-law word is as follows:

$$\overline{m0}\ \overline{m1}\ \overline{m2}\ \overline{m3}\ \overline{n0}\ \overline{n1}\ \overline{n2}\ \overline{s}$$

This μ-law word represents the following number:

$$Y = (-1)^S * [(16.5+M) * 2^N - 16.5]$$

where:

M = m3 m2 m1 m0 (M is 0 to 15)
N = n2 n1 n0 (N is 0 to 7)
* = multiplication
— = bar (complement)

A-law

The format of an 8-bit, A-law word is as follows:

$$\overline{m0}\ m1\ \overline{m2}\ m3\ \overline{n0}\ n1\ \overline{n2}\ \overline{s}$$

This A-law word represents the following number:

$$Y = (-1)^S * [(16.5+M) * 2^N] \qquad N = 1,2, \ldots, 7$$
$$Y = (-1)^S * (0.5 + M) * 2 \qquad N=0$$

where:

M = m3 m2 m1 m0 (M is 0 to 15)
N = n2 n1 n0 (N is 0 to 7)
* = multiplication
— = bar (complement)

8.2.4.3 8-bit Unsigned Integer

The DAU can perform conversions between 8-bit unsigned integers and DSP3210 floating-point values in a single instruction cycle. The 8-bit integer values are assumed to be in the range 0 to 255 when converting to floating-point. Floating-point values are converted to an 8-bit unsigned integer. If the floating-point operand exceeds the range of allowable 8-bit unsigned integers (0,255), a saturated 8-bit integer is returned (0xFF for the most positive or 0x00 for the least positive). Negative floating-point numbers are converted to 0x00. Based on the dauc register, floating-point numbers are converted to 8-bit unsigned numbers by rounding or truncation (See 8.2.5 Rounding Modes). If the 8-bit integer result is written to memory, it is loaded into an 8-bit memory location.

8.2.4.4 16-bit Integer

The DAU supports conversion from a floating-point operand to a 16-bit 2's complement integer. If the 16-bit integer result is written to memory, it is loaded into a 16-bit memory location. If the floating-point operand exceeds the range of allowable 16-bit integers (-2^{15}, $2^{15} - 1$), a saturated 16-bit integer is returned (0x8000 or 0x7fff, respectively). Based on the dauc register, floating-point numbers are converted to 16-bit linear by rounding or truncation (See 8.2.5 Rounding Modes). Similarly, the DAU supports conversion from a 16-bit integer to the equivalent 32-bit floating-point representation.

8.2.4.5 32-bit Integer

The DAU supports conversion from a floating-point operand to a 32-bit 2's complement integer. If the 32-bit integer result is written to memory, it is loaded into a 32-bit memory location. If the floating-point operand exceeds the range of allowable 32-bit integers (-2^{31}, $2^{31} - 1$), a saturated 32-bit integer is returned (0x80000000 or 0x7fffffff, respectively). Based on the dauc register, floating-point numbers are converted to 32-bit linear by rounding or truncation (See 8.2.5 Rounding Modes). Similarly, the DAU supports conversion from a 32-bit integer to the equivalent 32-bit floating-point representation.

8.2.4.6 Reciprocal Seed

In the DSP3210, a divide operation can be performed by calculating the reciprocal of the divisor, and multiplying this value by the dividend. An instruction is provided to convert, in hardware, a 32-bit floating-point number to a 32-bit seed value as a first approximation to the reciprocal of the number. This one-cycle instruction produces a seed with a mantissa accurate to 3 bits.

8.2.5 Rounding Modes

On conversions from floating-point to 32-bit or 16-bit signed integers, or to 8-bit unsigned integers, the dauc register (see Table 8-3) controls whether the integer result is rounded to nearest, truncated towards -∞, or truncated towards zero. Figure 8-11 graphically displays the differences between the three modes.

With round-to-nearest, the resulting integer is incremented if there is a one in the most significant fraction bit of the denormalized floating-point number. Note that this rounds 0.5, 1.5, 2.5,... to 1, 2, 3,..., respectively and –0.5, –1.5, –2.5,... to 0, –1, –2,..., respectively.

With truncate towards -∞, the fraction bits of the denormalized floating-point number are discarded.

With truncate towards zero, the fraction bits of the denormalized floating-point number are treated differently based on whether the number is positive or negative. For positive numbers, the fraction bits are discarded. For negative numbers, the integer is incremented by one if there were any 1s in the fraction bits.

On conversion of 8-bit unsigned integers, there is no difference between truncate towards zero and truncate towards -∞ because all negative numbers are saturated to zero.

Note that the *round()* instruction rounds a 40-bit floating-point number to a 32-bit floating-point number, and it is always performed as round-to-nearest, regardless of the dauc setting.

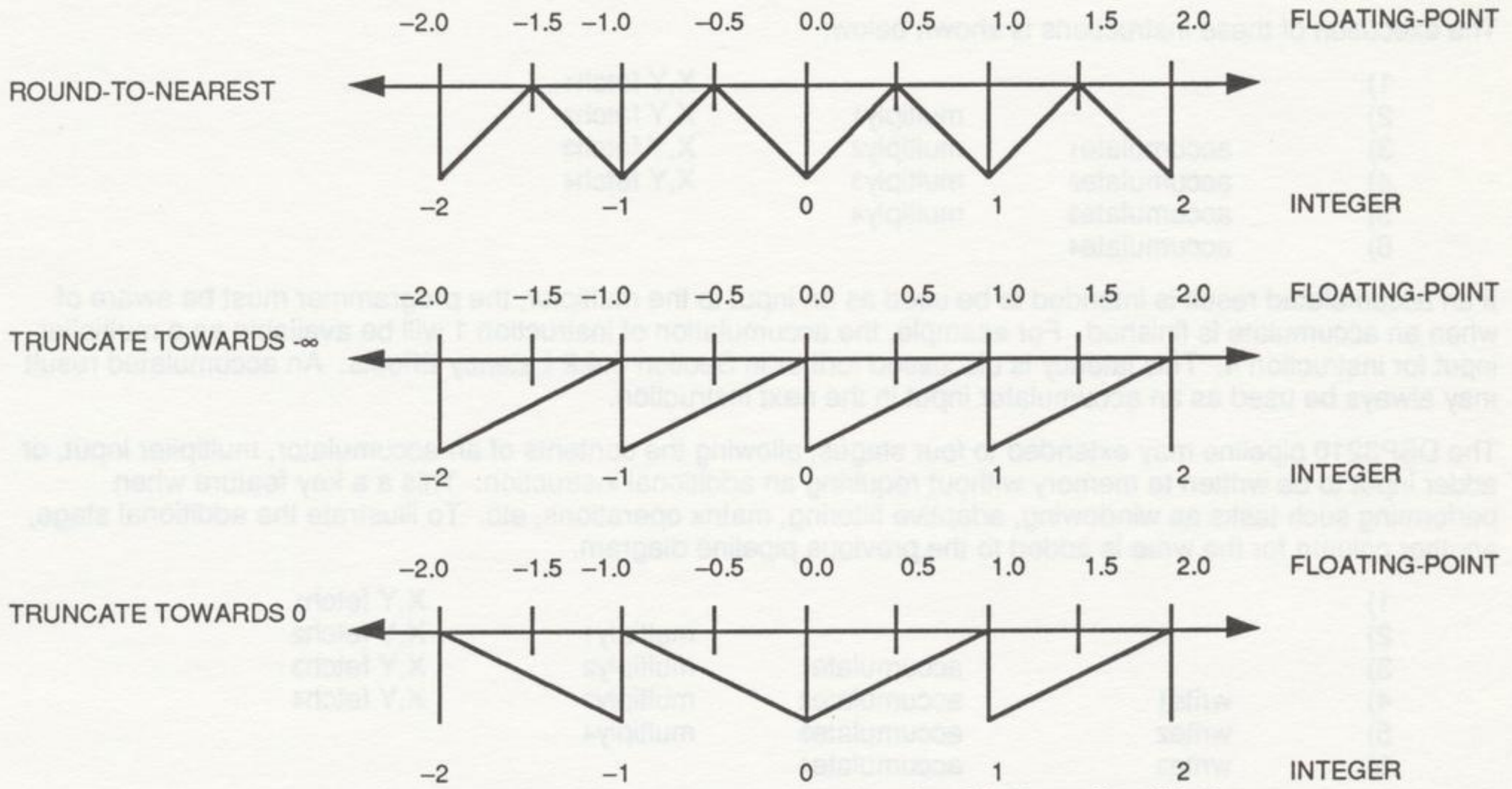


Figure 8-11 Floating-Point to Integer Mapping for All Rounding Modes

8.2.6 DAU Internal Operation

The following section presents a description of a single instruction as it passes through the DAU pipeline. The DAU supports four multiply-accumulate instruction formats, providing flexibility in choosing operand and storing results. To simplify this discussion, one format will be discussed. The DSP3210 executes the multiply-accumulate instruction

aN = aM + Y * X

in three stages (an optional fourth stage to perform a write to memory is discussed later); fetch of X and Y, multiplication of X and Y, and accumulation of the Y * X product with the contents of an accumulator, aM. This instruction executes as follows:

step 1) X,Y fetch
step 2) multiply Y * X
step 3) accumulate product with aM

If several multiply-accumulate instruction are executed one after the other, the DSP3210 automatically pipelines the instructions so that one instruction is completed every instruction cycle. This is shown in the following block of instructions where X_1-X_4 and Y_1-Y_4 are subscripted to make it easier to follow the data flow in the DAU.

1) $aN = aM + Y_1 * X_1$
2) $aN = aM + Y_2 * X_2$
3) $aN = aM + Y_3 * X_3$
4) $aN = aM + Y_4 * X_4$

The execution of these instructions is shown below:

1)			X,Y fetch1
2)		multiply1	X,Y fetch2
3)	accumulate1	multiply2	X,Y fetch3
4)	accumulate2	multiply3	X,Y fetch4
5)	accumulate3	multiply4	
6)	accumulate4		

If an accumulated result is intended to be used as an input to the multipier, the programmer must be aware of when an accumulate is finished. For example, the accumulation of instruction 1 will be available as a multiplier input for instruction 4. This latency is discussed further in Section 4.4.2 Latency Effects. An accumulated result may always be used as an accumulator input in the next instruction.

The DSP3210 pipeline may extended to four stages, allowing the contents of an accumulator, multiplier input, or adder input to be written to memory without requiring an additional instruction. This a a key feature when performing such tasks as windowing, adaptive filtering, matrix operations, etc. To illustrate the additional stage, another column for the write is added to the previous pipeline diagram.

1)				X,Y fetch1
2)			multiply1	X,Y fetch2
3)		accumulate1	multiply2	X,Y fetch3
4)	write1	accumulate2	multiply3	X,Y fetch4
5)	write2	accumulate3	multiply4	
6)	write3	accumulate4		

The syntax of such an instruction is

Z = aN = aM + Y * X

Note that the memory location written by instruction 1 will be available as multiplier input in instruction 5.

The internal pipeline timing of the DAU for a single instruction of the form Z = aN = aM + Y * X is shown in Figure 8-12. An instruction requires five instruction cycles (I_0—I_4) to be decoded and executed in the DAU. The instruction (I) appears on the data bus in state 3 preceding I_0. The X and Y registers are loaded in states 1 and 2 of I_1, respectively. The contents of these registers are multiplied during I_2 and the product is loaded into the product register, P, in state 3 of I_2. The adder input register, S, is also loaded in state 3 of I_2. The contents of the P and S registers are added during I_3 and the result is loaded into an accumulator in state 3 of I_3. This same result is then placed on the data bus (which is routed to a destination in memory) in state 0 of I_4, if a Z-field is specified in the instruction. The Z-field is optional.

The DAU supports two multiply-accumulate instruction formats called tap-update instructions. In one format,

aN = aM + (Z = Y) * X

instead of the result of the multiply-accumulate operation being written to memory, the Y operand.is written. It must be written in the same cycle that the multiply-accumulate result would have been written (state 0 of I4). Because the Y operand appears on the data bus in state 2 of I1, it must be latched and held for three instruction cycles before being written to memory. This three instruction cycle delay is performed in the tap-update delay block in Figure 8-7. The other tap-update instruction has the form:

aN = (Z = Y) + X.

The four-stage pipeline in the DAU contributes to the high throughput of the DSP3210.

	Cycle 0					Cycle 1				Cycle 2				Cycle 3				Cycle 4			
State	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
RAM0					X Fetch																
RAM1						Y Fetch														Z Store	
Data Bus	I						X	Y										Z			
X							X														
Y								Y													
Multiplier										Multiply X * Y											
P													P								
S													aM								
Adder														Add S + P							
aN (a0—a3)																	aN				

For instructions of the form: Z = aN = aM + Y * X,

where X, Y = data reference (memory, a0—a3)
aM = a0-a3
aN = a0-a3
Z = data write (memory)
P= product
S= adder input

Figure 8-12 DAU Internal Pipeline Timing

Chapter 9

Peripheral Functional Units

CHAPTER 9. PERIPHERAL FUNCTIONAL UNITS

9. PERIPHERAL FUNCTIONAL UNITS 9-1
9.1 Serial I/O 9-1
9.1.1 SIO Programming Model 9-1
Figure 9-1 SIO Block Diagram 9-3
Table 9-1 ioc Register 9-4
Table 9-2 Pin Descriptions 9-5
9.1.2 Pin Descriptions 9-5
9.1.3 Serial Input Port 9-5
9.1.3.1 Configuring the Serial Input Port 9-6
Figure 9-2 Serial Input Port 9-7
9.1.3.2 Serial Input Timing Specifications 9-7
Figure 9-3 Serial Input Protocols (IIC = 0) 9-8
Figure 9-4 Serial Input Timing (IIC = 0) 9-8
9.1.4 Serial Output Port 9-9
Figure 9-5 Serial Output Port 9-9
9.1.4.1 Configuring the Serial Output Port 9-10
Figure 9-6 Serial Output Protocol 9-10
9.1.4.2 Serial Output Timing Specifications 9-10
Figure 9-7 Serial Output Timing 9-11
Figure 9-8 On-Chip Clock Generator 9-11
9.1.5 SIO On-Chip Clock Generator 9-12
9.1.5.1 Configuring the Clock Generator 9-12
9.1.5.2 Clock Generator Timing Specifications 9-12
9.1.5.3 Loss of Sanity 9-13
9.1.5.4 Loss of Sync 9-13
Figure 9-9 Clock Generator Signals 9-13
Figure 9-10 Clock Generation Timing (IIC = 0) 9-14
9.1.5.5 Frame Boundary Condition 9-14
Figure 9-11 Frame Boundary Condition 9-15
9.1.6 Handling SIO activity 9-15
9.1.6.1 Programmed Controlled SIO 9-15
9.1.6.2 Interrupt Controlled SIO 9-16
Listing 9-1 9-16
9.1.6.3 DMA Controlled SIO 9-17
Listing 9-2 Input DMA Example 9-17
9.2 DMA Controller (DMAC) 9-18
9.2.1 DMAC Programming Model 9-18
9.2.2 DMAC Functional Description 9-18
Figure 9-12 DMAC Block Diagram 9-19
9.2.3 Serial Output DMA 9-19
9.2.4 Serial Input DMA 9-20
Table 9-3 DMAC Register Encoding 9-20
9.3 Timer 9-21
9.3.1 Timer Programming Model 9-21
9.3.2 Timer Functional Description 9-21
Figure 9-13 Timer Block Diagram 9-22
Table 9-4 TCON Register Encoding 9-23
9.3.3 Timer Timing Specifications 9-24
Figure 9-14 Timer Timing Specifications 9-24
9.4 Bit I/O (BIO) 9-24
9.4.1 BIO Programming Model 9-25
Table 9-5 bioc Register Encoding 9-25
Table 9-6 BIO Register Encodings (Writing and Reading) 9-25
Figure 9-15 Bit I/O Bit Slice Diagrams 9-26
9.4.2 BIO Functional Description 9-26
9.4.3 BIO Timing Specifications 9-26
Figure 9-16 BIO Timing Specifications 9-27

9. PERIPHERAL FUNCTIONAL UNITS

The DSP3210 provides four on-chip peripheral units; serial I/O (SIO), DMA controller (DMAC), 32-bit timer (TIMER), and bit I/O (BIO). Access to peripheral functional units is memory-mapped. Detailed descriptions of these facilities are contained in this section.

9.1 Serial I/O

The serial I/O (SIO) section provides synchronous, double-buffered serial ports for concurrent input and output of digital data. Inputs and outputs can be handled under program control, DMA mode, or interrupt mode. The serial I/O can interface to a TDM (Time Division Multiplexed) line, serial codecs, or other DSPs (DSP32, DSP32C, DSP3210, DSP16, DSP16A) with few, if any, additional components.

The SIO is divided into three functional units, the serial input port (SIP), the serial output port (SOP), and the SIO clock generator (see Figure 9-1). There is a register, ioc, that allows the programmer to configure these functional units for various modes of operation. Registers that are accessible to the programmer are denoted by names in lower-case.

The description of the SIO is divided into five sections: SIO programming model, pin descriptions, serial input port, serial output port, and on-chip SIO clock generator. This chapter also includes a section on special purpose SIO flags and error condition reporting. The remainder of the SIO section describes programmed, DMA, and interrupt modes of handling serial I/O activity.

9.1.1 SIO Programming Model

The programming model for the SIO consists of three MMIO registers: ibuf (MMIO[0x0404]), obuf (MMIO[0x0408]), and ioc (MMIO[0x040C]). The MMIO address corresponds to the 16 LSB's of the 32-bit address. The address is given for both big and little endian byte orderings. The byte order is specified by pcw[8]. The upper 16-bits are a function of the memory mode configured in the pcw register. The register programming model for the SIO is pictured below:

Register Name	Address Big Endian	Address LIttle Endian	Register Bits 31 24 23 16 15 8 7 0	
ibuf	0x0404*	0x0404*		R
obuf	0x0408†	0x0408†		W
ioc	0x040C	0x040C		R/W

* Shows 32-bit read address.
† Shows 32-bit write address.

The ioc (input/output control) register is a 32-bit read-write register that configures the serial input port, the serial output port, and the on-chip SIO clock generator. The position, mnemonic, encoding, and the definition of each bit field of the ioc register are described in Table 9-1. The serial input buffer, ibuf, may be read as an 8-bit, 16-bit, or 32-bit register. Referencing ibuf is a function of bit mode (least significant bit first/most significant bit first), byte ordering, and data size. When ibuf is used as an 8-bit or 16-bit register, valid data is read as follows:

Mode	Byte Ordering	Data Size	Register Bits	Address
LSB	BE	8	31—24	0x0404
	LE	8	31—24	0x0407
MSB	BE	8	7—0	0x0407
	LE	8	7—0	0x0404
LSB	BE	16	31—16	0x0404
	LE	16	31—16	0x0406
MSB	BE	16	15—0	0x0406
	LE	16	15—0	0x0404

Referencing obuf is a function of byte ordering and data size. When obuf is used an an 8-bit or 16-bit register, valid data is written as follows:

Mode	Byte Ordering	Data Size	Register Bits	Address
MSB/MSB	BE	8	15—0	0x040a
	LE	8	15—0	0x0408
LSB/MSB	BE	8	7—0	0x040b
	LE	8	7—0	0x0408

The serial output buffer, obuf, may be written as an 8-bit, 16-bit, or 32-bit register. In the remainder of this chapter, the mnemonics ibuf, obuf, and ioc will be used exclusively and not the MMIO address. These registers may be read or written using any instruction supporting memory references. The DMA controller uses the ibuf and obuf as source and destination registers, respectively, for DMA transfers. Reset clears the ioc register to all zeros, but has no effect on the ibuf and obuf registers.

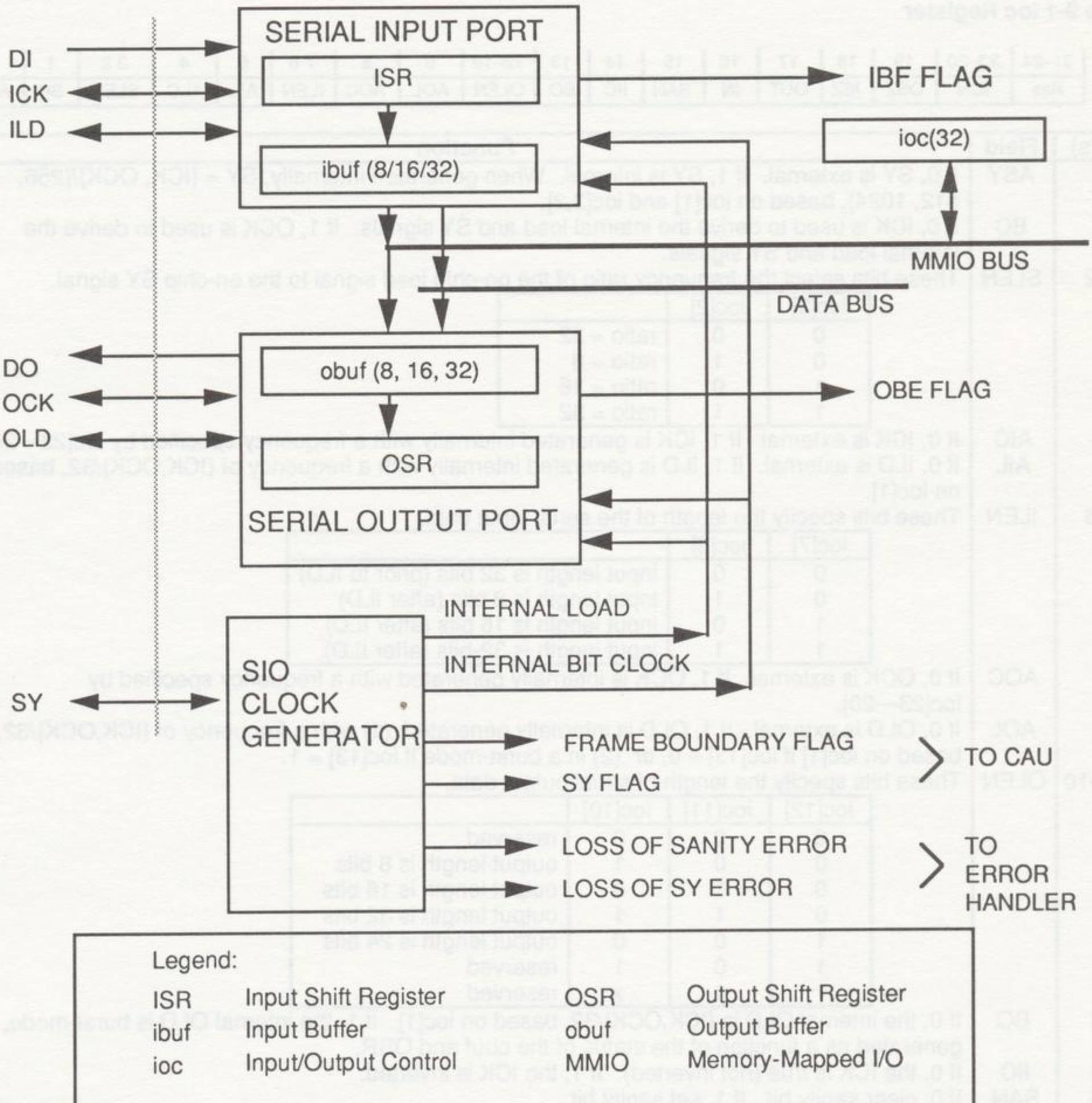


Figure 9-1 SIO Block Diagram

Table 9-1 ioc Register

Bit	31–24	23–20	19	18	17	16	15	14	13	12–10	9	8	7 6	5	4	3 2	1	0
Field	Res	ICN	OSZ	ISZ	OUT	IN	SAN	IIC	BO	OLEN	AOL	AOC	ILEN	AIL	ALC	SLEN	BC	ASY

Bit(s)	Field	Function
0	ASY	If 0, SY is external. If 1, SY is internal. When generated internally, SY = {ICK, OCK}/{256, 512, 1024}, based on ioc[1] and ioc[3,2].
1	BC	If 0, ICK is used to derive the internal load and SY signals. If 1, OCK is used to derive the internal load and SY signals.
3,2	SLEN	These bits select the frequency ratio of the on-chip load signal to the on-chip SY signal.
4	AIC	If 0, ICK is external. If 1, ICK is generated internally with a frequency specified by ioc[23—20].
5	AIL	If 0, ILD is external. If 1, ILD is generated internally with a frequency of {ICK,OCK}/32, based on ioc[1].
7,6	ILEN	These bits specify the length of the serial input data.
8	AOC	If 0, OCK is external. If 1, OCK is internally generated with a frequency specified by ioc[23—20].
9	AOL	If 0, OLD is external. If 1, OLD is internally generated: (1) with a frequency of {ICK,OCK}/32, based on ioc[1] if ioc[13] = 0, or (2) in a burst-mode if ioc[13] = 1.
12—10	OLEN	These bits specify the length of serial output data.
13	BO	If 0, the internal OLD is {ICK,OCK}/32, based on ioc[1]. If 1, the internal OLD is burst-mode, generated as a function of the status of the obuf and OSR.
14	IIC	If 0, the ICK is true (not inverted). If 1, the ICK is inverted.
15	SAN	If 0, clear sanity bit. If 1, set sanity bit.
16	IN	If 0, the LSB is received first during serial inputs. If 1, the MSB is received first during serial inputs.
17	OUT	If 0, the LSB is transmitted first during serial outputs. If 1, the MSB is transmitted first during serial outputs (can't be used with 24 bit output length).
18	ISZ	If 0, the input DMA transfer size is 32-bits. If 1, the input DMA size is specified by ILEN.
19	OSZ	If 0, the output DMA transfer size is 32-bits. If 1, the output DMA size is specified by OLEN.
23—20	ICN	These bits specify the divide-down rate of the internal clock generator. The internal clock generated has an additional divide-down by four, i.e., if bits 23—20 represent N, the clock rate is N*4. If ICN=0, the active clock generator is disabled.
31—24		Reserved; Read as zero.

ioc[3]	ioc[2]	
0	0	ratio = 32
0	1	ratio = 8
1	0	ratio = 16
1	1	ratio = 32

ioc[7]	ioc[6]	
0	0	input length is 32 bits (prior to ILD)
0	1	input length is 8 bits (after ILD)
1	0	input length is 16 bits (after ILD)
1	1	input length is 32-bits (after ILD)

ioc[12]	ioc[11]	ioc[10]	
0	0	0	reserved
0	0	1	output length is 8 bits
0	1	0	output length is 16 bits
0	1	1	output length is 32 bits
1	0	0	output length is 24 bits
1	0	1	reserved
1	1	x	reserved

Table 9-2 Pin Descriptions

Pin	Name	Description
DI	Data Input	Input for serial digital data. Data is sampled with the rising edge of ICK if IIC = 0, or the falling edge if IIC = 1.
ICK	Input Clock	Clock for serial digital input data. Serial input port activity is based on ICK. Depending on the ioc setting, this signal may be either an input (external mode) or an output (internal mode).
ILD	Input Load	A high-to-low transition on this pin synchronizes the loading of the input buffer from the input shift register. Depending on the ioc setting, the signal can be either an input (external mode) or an output (internal mode).
DO	Data Output	Output for serial digital data. Data is shifted with the rising edge of OCK.
OCK	Output Clock	Clock for serial digital output data. Serial output port activity is based on the rising edge of OCK. Depending on the ioc setting, this signal may be either an input (external mode) or an output (internal mode).
OLD	Output Load	A high-to-low transition on this pin causes the loading of the output shift register from the output buffer. Depending on the ioc setting, the signal can be either an input (external mode) or an output (internal mode).
SY	Synchronization	Provides frame synchronization either to the DSP3210 from peripheral components (external mode) or from the DSP3210 to peripheral components (internal mode), depending on the ioc setting. The logical state of this pin may be tested by CA instructions (see Section 4.3 CA Instructions).

9.1.2 Pin Descriptions

The SIO has seven pins. Three pins are associated with the SIP, three with the SOP, and one with the SIO clock generator. Two pins in each of the input and output sections (corresponding to the bit clock and load signal) and the pin for the SIO clock generator (corresponding to the frame signal) are bidirectional. These signals are denoted as **internal** and are an output if they are generated by the DSP3210. These signals are denoted as **external** and are an input if they are supplied from an outside source. Table 9-2 is a description of the seven pins associated with the serial I/O. For each pin, the pin name, full name, and a description of its function are given.

9.1.3 Serial Input Port

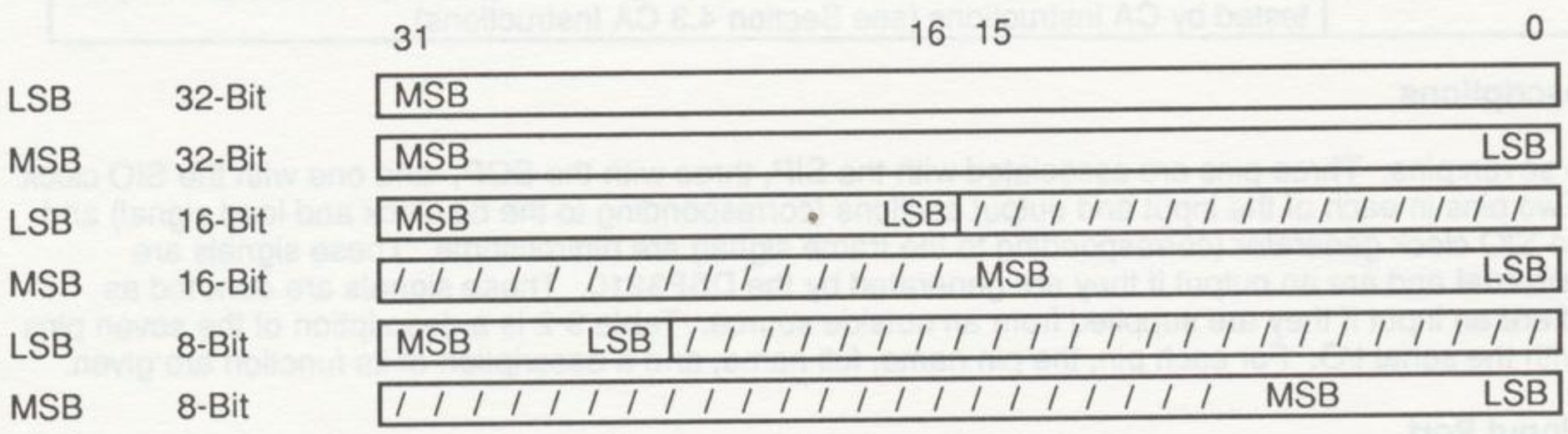
The serial input port (SIP) is used by the DSP3210 to receive serial data from external devices or data streams. The serial input port consists of the input buffer (ibuf), the input shift register (ISR), the input load shift register (ILSR), and associated logic, as shown in Figure 9-2. The SIP supports a synchronous three wire protocol: clock (ICK), load (ILD), and data (DI). An internal flag, input buffer full (IBF), is also available to indicate the status of the ibuf.

The edge of the clock used to clock data in the SIP is configurable by ioc[14], IIC. When IIC=0, the serial input port is sensitive to the rising edge of ICK. When IIC=1, ICK is inverted on-chip making the port sensitive to the falling edge of ICK. Serial data in (DI) and the load signal (ILD) are clocked into the ISR and ILSR, respectively. The load signal is delayed in the ILSR at the bit clock rate and is used to generate a signal to load the ibuf from the ISR. The number of clocks that the load signal is delayed is a function of the input length (ILEN) specified in the ioc register. When the IBUF is loaded, the input buffer full (IBF) flag is set high. When the IBUF is read by a DSP3210 program, interrupt routine, or DMA facility, the IBF flag is set low. Together, the ISR and IBUF form a double buffer so data can be shifted into the ISR while data is waiting to be read from the IBUF.

9.1.3.1 Configuring the Serial Input Port

ICK and ILD can be supplied off-chip (external mode), from an external peripheral, or on-chip (internal mode) from the on-chip SIO clock generator (see 9.1.5 SIO On-chip Clock Generator). The selections of internal or external mode for ICK, ioc[4], and ILD, ioc[5], are totally independent of each other. The internal clock generator, as configured by ioc[3-0] and ioc[23—20], generates the internal bit clock, load signal, and sync signal. The internal bit clock rate can be set for rates between CKI/4 and CKI/60 via ioc[23—20], ICN. The internal load signal is a divide by 32 of either ICK or OCK, based on ioc[1]. Note that with a continuous bit clock, the internal load will occur every 32 periods of the bit clock. The ICK can be configured to be true or inverted by ioc[14], IIC. When ICK is true, the SIP is sensitive to the rising edge of ICK. When ICK is inverted, the SIP is sensitive to the falling edge of ICK.

The length of the input data can be specified to be 8, 16 or 32 bits via ioc[6,7], ILEN. If ILEN is set to 00, then the input data corresponds to the 32 bits **prior** to the ILD high-to-low transition. This mode can be used for data lengths other than 8, 16, and 32 bits by masking off the undesired bits. For the 01,10, and 11 cases, the input data corresponds to the 8, 16, or 32 bits **after** the ILD high-to-low transition, respectively. These two protocols for serial input are shown in Figure 9-3 (IIC = 0, positive edge-triggered, is shown). The order of the serial input data bits can be specified as least significant bit first, LSB, or most significant bit first, MSB, via ioc[16], IN. IN is used to control the direction of the bidirectional shift register ISR and select which bytes of the IBUF are put on the internal Data or MMIO Bus when the input buffer is read. Input data resides in the ibuf register bits as a function of LSB or MSB mode and the size specified. Valid data in the input buffer is organized as follows:



The Data Bus is used for DMA controlled transfers and the MMIO bus is used for programmed transfers.

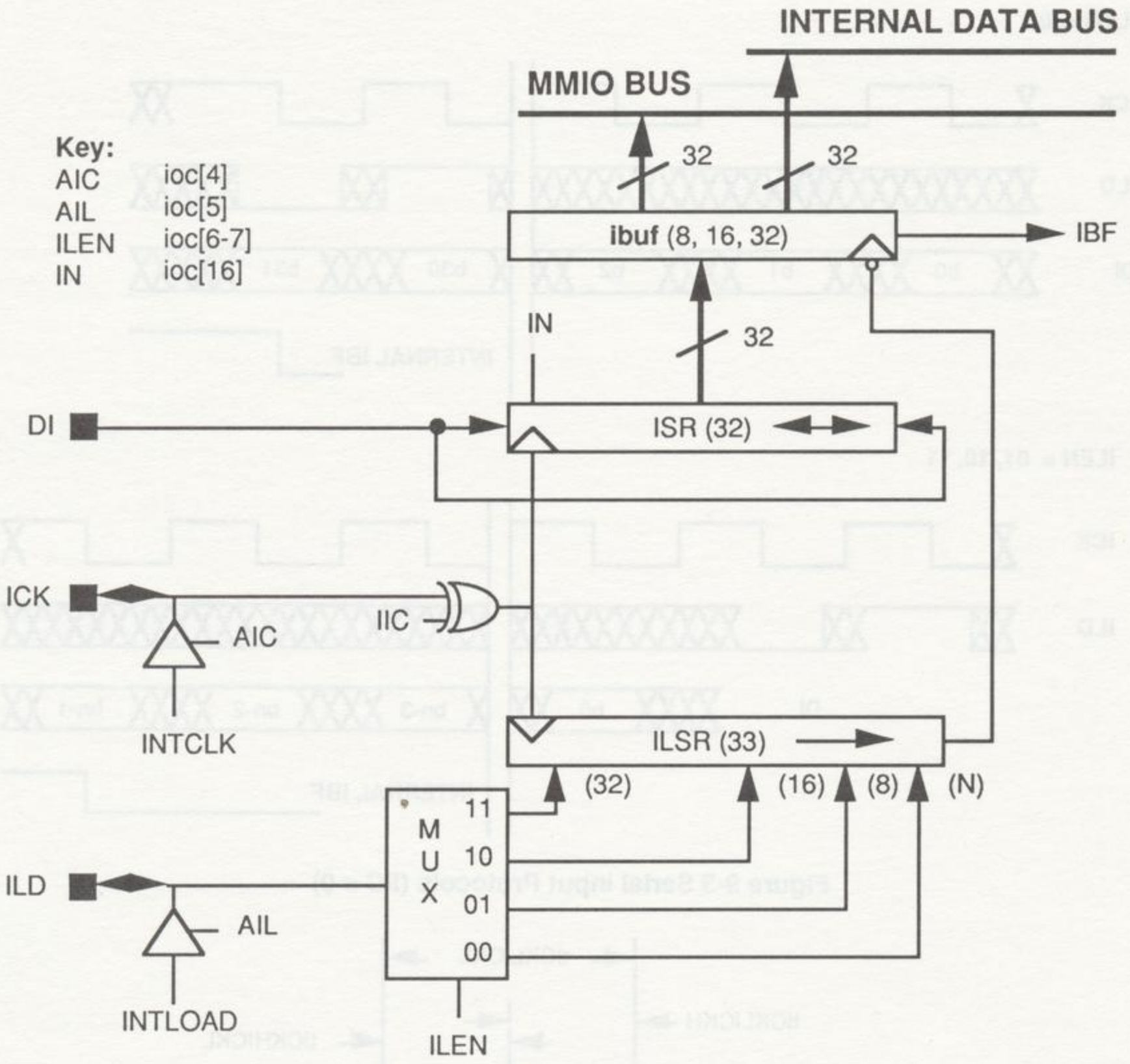


Figure 9-2 Serial Input Port

9.1.3.2 Serial Input Timing Specifications

The timing requirements and characteristics for serial input transmissions are shown in Figure 9-4 (IIC = 0, not inverted). The specifications are described as follows:

- **ICK Clock Requirements:** The clock requirements are the period of the input data bit clock ICK (tICKLICKL), the high time of ICK (tICKHICKL), and the low time of ICK (tICKLICKH).
- **ILD Requirements to Initiate an Input Transmission:** ILD is set up high before the rising edge of ICK (tILDHICKH), and is held high until after the rising edge of ICK (tICKHILDL). ILD is set up low before the next rising edge of ICK (tILDLICKH) and is held low until after that edge of ICK (tICKHILDH). Similar requirements must be met when IIC=1 (negative edge-triggered).
- **DI Requirements:** Serial input data bits 0 through n-1 must satisfy set up (tDIVICKH) and hold (tICKHDIX) times to be latched and shifted in the ISR on the rising edge of ICK. Similar requirements must be met when IIC = 1 (negative edge-triggered).

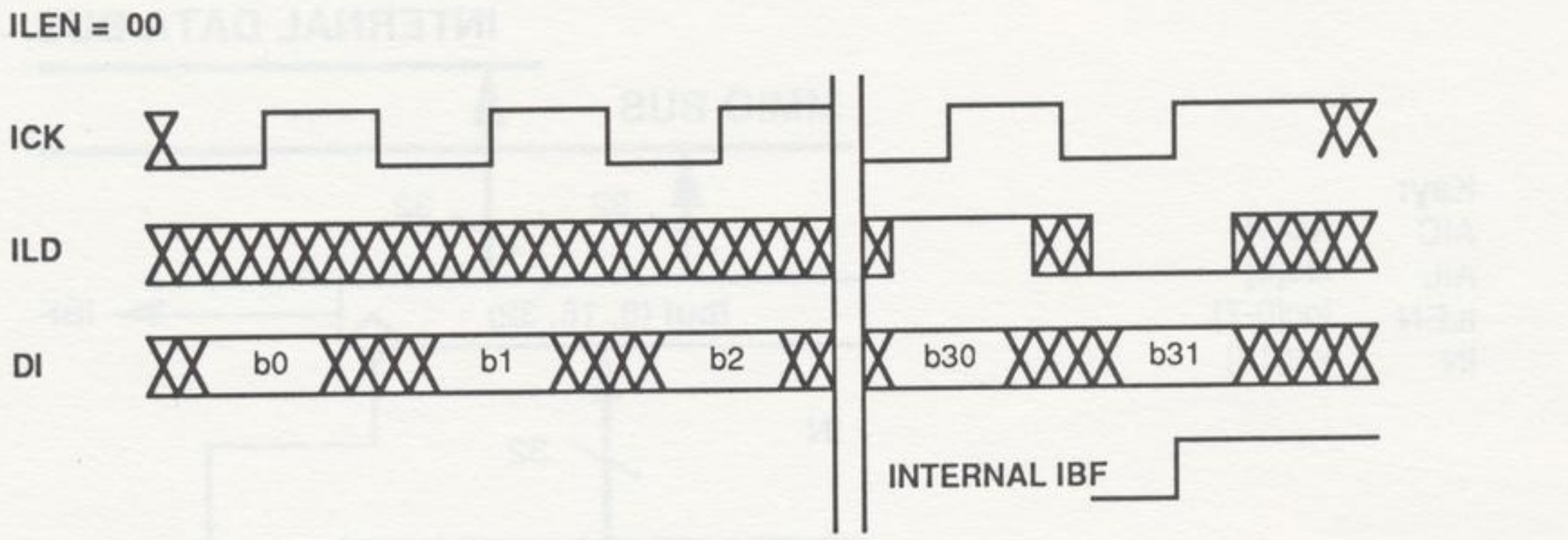


Figure 9-3 Serial Input Protocols (IIC = 0)

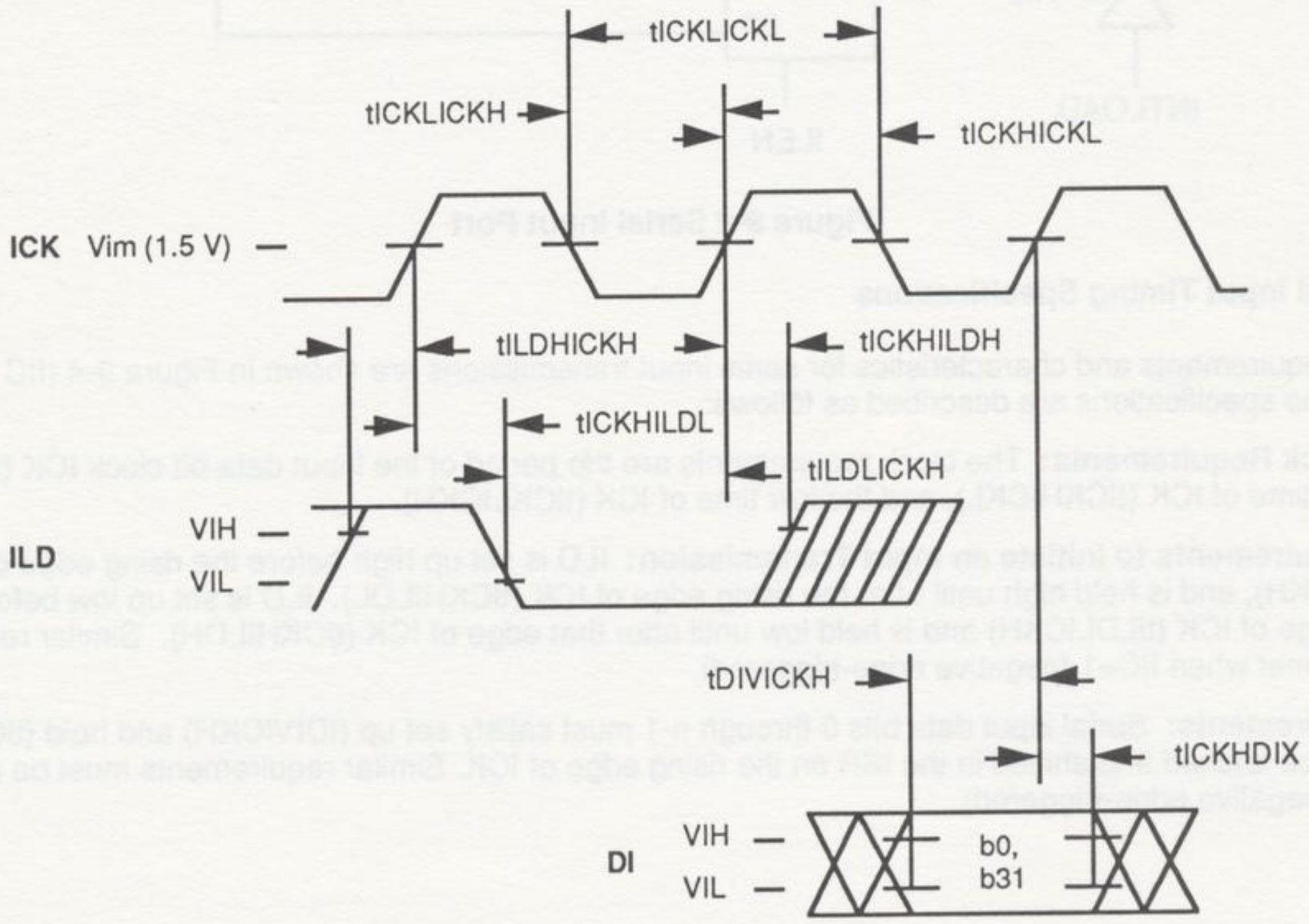


Figure 9-4 Serial Input Timing (IIC = 0)

9.1.4 Serial Output Port

The serial output port (SOP) is used to transmit serial data from the DSP3210 to external devices or data streams. The serial output port consists of the output buffer (obuf), the output shift register (OSR), the output load shift register (OLSR), and associated logic, as shown in Figure 9-5. The SOP supports a synchronous three-wire protocol: clock (OCK), load (OLD), and data (DO). An internal flag, output buffer empty (OBE), is available to indicate the status of the OBUF.

Serial output data on DO is clocked out of the output shift register (OSR) on the rising edge of the clock OCK. When the obuf is loaded by the program, the interrupt routine, or the DMA facility, the output buffer empty (OBE) flag is set low. When the data in obuf is transferred to the OSR, the OBE flag is set high. Together, the obuf and OSR form a double buffer so that data can be shifted out of the OSR while data is loaded into the obuf. On each rising edge of OCK, the load signal (OLD) is clocked and shifted in the output load shift register (OLSR). The load signal is delayed by the OLSR at the bit clock rate and is used in the generation of the load/shift signal for the OSR and the internal burst OLD (INTLOADB). If selected by ioc[13] and ioc[9], a burst OLD signal can be generated by the SIO that is active only when new data has been loaded into the obuf rather than on a periodic basis.

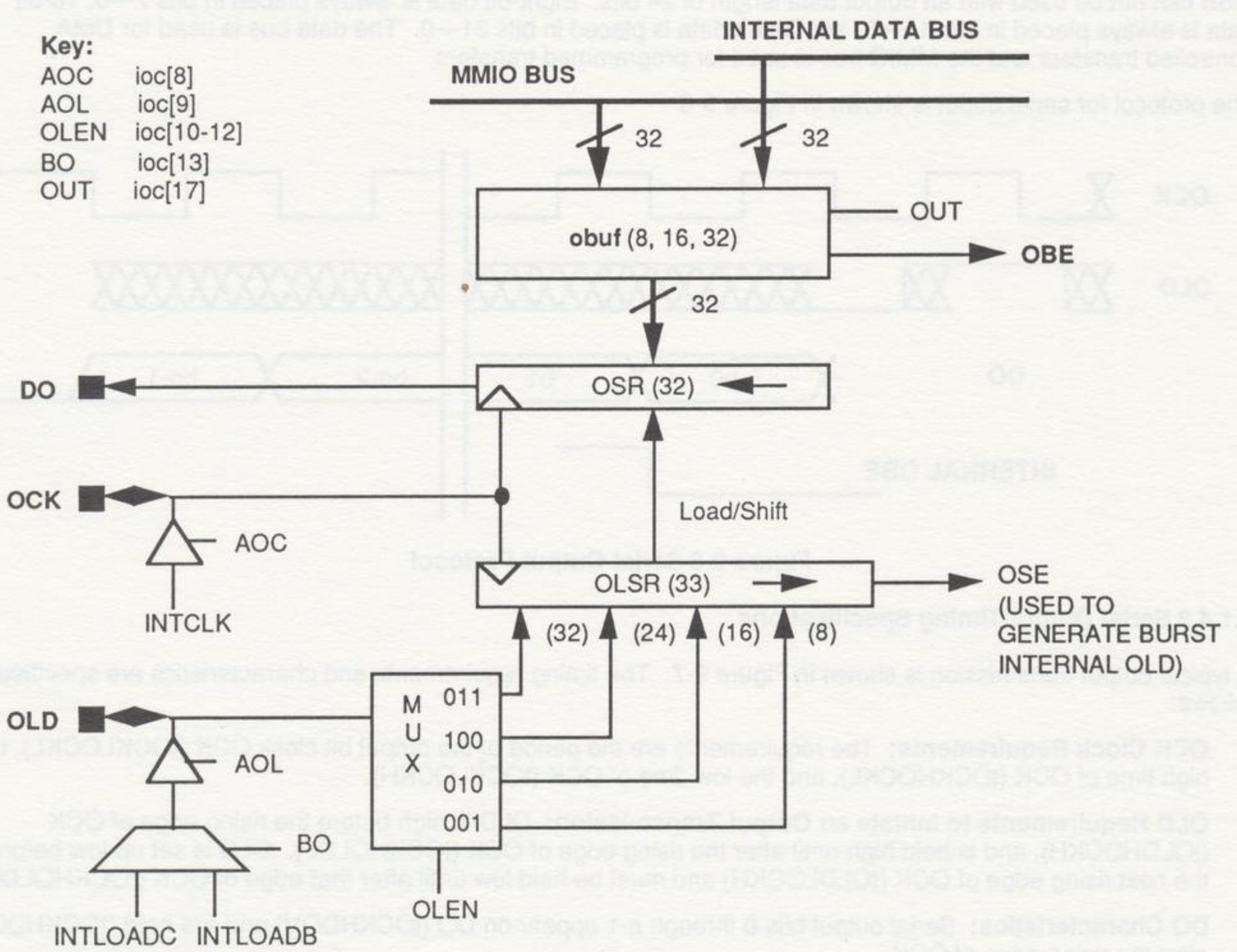


Figure 9-5 Serial Output Port

9.1.4.1 Configuring the Serial Output Port

OCK and OLD can be supplied off-chip (external mode) from an external peripheral, or on-chip (internal mode) from the on-chip SIO clock generator (see Section 9.1.5 SIO On-chip Clock Generator). The selections of internal or external mode for OCK, ioc[8], and OLD, ioc[9], are totally independent of each other. The internal clock generator, as configured by ioc[3—0] and ioc[23—20], generates the internal bit clock and load signal. The internal bit clock rate can be set for rates between CKI/4 and CKI/60 via ioc[23—20], ICN. The internal load signal can be configured in two ways: (1) continuous or (2) burst. The continuous internal load signal is a divide by 32 of either ICK or OCK, based on ioc[1]. Note that with a continuous bit clock, the internal load signal will occur every 32 bit clocks. The burst OLD signal uses the OBE flag and the OLSR output (OSE) to generate an OLD that is active only when new data has been loaded into the OBUF.

The length of output data can be specified to be 8, 16, 24, or 32 bits via ioc[12—10]. In all cases, the data corresponds to the bits after the OLD high-to-low transition. The order of the serial output bits can be specified as least significant bit first, LSB, or most significant bit first, MSB, via ioc[17], OUT. OUT is used to select which bit of the Internal Data Bus or MMIO Bus loaded into each bit of OBUF. If OUT is set to 0, bit 0 → obuf(0), bit 1 → obuf(1), ..., bit 31 → obuf(31). If OUT is set to 1, bit 0 → obuf(31), bit 1 → obuf(30), ..., bit 31 → obuf(0). MSB can not be used with an output data length of 24 bits. Eight-bit data is always placed in bits 7—0, 16-bit data is always placed in bits 15—0, and 32-bit data is placed in bits 31—0. The data bus is used for DMA controlled transfers and the MMIO bus is used for programmed transfers.

The protocol for serial ouput is shown in Figure 9-6.

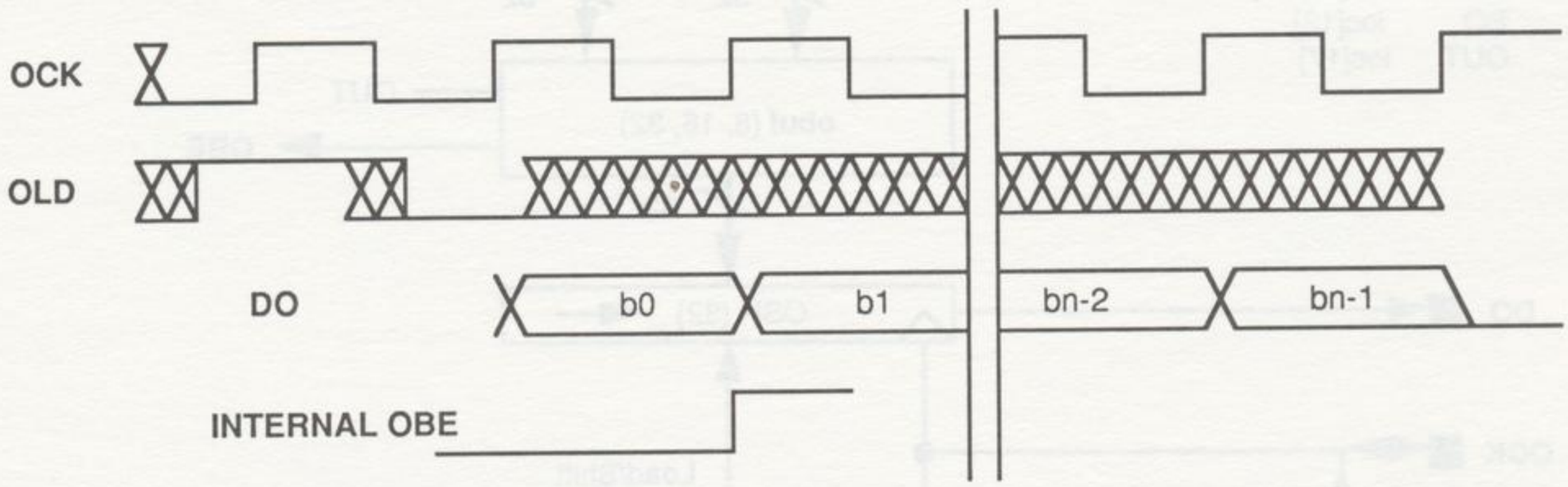


Figure 9-6 Serial Output Protocol

9.1.4.2 Serial Output Timing Specifications

A typical output transmission is shown in Figure 9-7. The timing requirements and characteristics are specified as follows:

- **OCK Clock Requirements:** The requirements are the period of the output bit clock OCK (tOCKLOCKL), the high time of OCK (tOCKHOCKL), and the low time of OCK (tOCKLOCKH).
- **OLD Requirements to Initiate an Output Transmission:** OLD is high before the rising edge of OCK (tOLDHOCKH), and is held high until after the rising edge of OCK (tOCKHOLDL). OLD is set up low before the next rising edge of OCK (tOLDLOCKH) and must be held low until after that edge of OCK (tOCKHOLDH).
- **DO Characteristics:** Serial output bits 0 through n-1 appear on DO (tOCKHDOV) and are held (tOCKHDOX) after the rising edge of OCK.

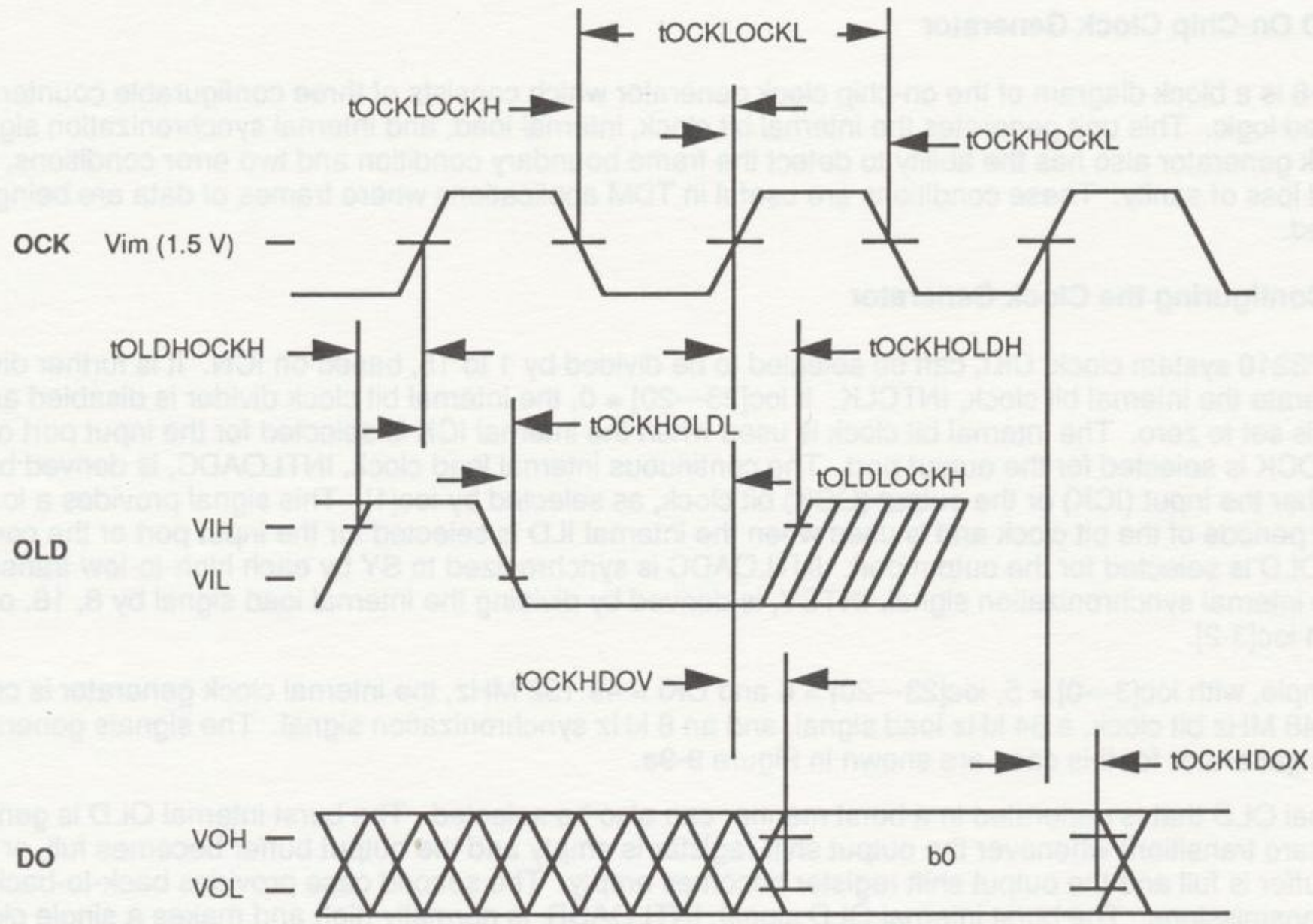


Figure 9-7 Serial Output Timing

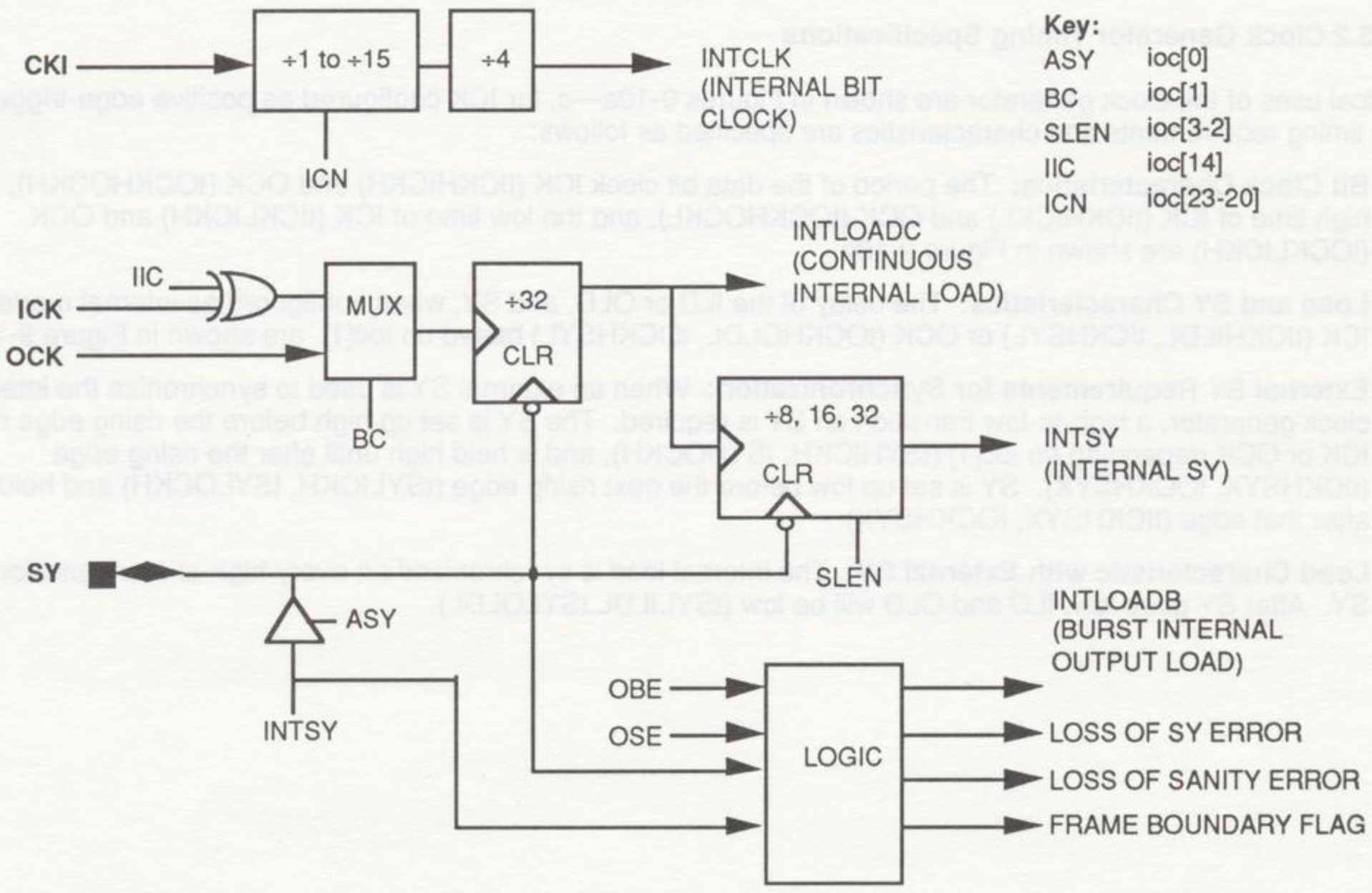


Figure 9-8 On-Chip Clock Generator

9.1.5 SIO On-Chip Clock Generator

Figure 9-8 is a block diagram of the on-chip clock generator which consists of three configurable counters and associated logic. This unit generates the internal bit clock, internal load, and internal synchronization signals. The clock generator also has the ability to detect the frame boundary condition and two error conditions, loss of sync and loss of sanity. These conditions are useful in TDM applications where frames of data are being processed.

9.1.5.1 Configuring the Clock Generator

The DSP3210 system clock, CKI, can be selected to be divided by 1 to 15, based on ICN. It is further divided by 4 to generate the internal bit clock, INTCLK. If ioc[23—20] = 0, the internal bit clock divider is disabled and INTCLK is set to zero. The internal bit clock is used when the internal ICK is selected for the input port or the internal OCK is selected for the output port. The continuous internal load clock, INTLOADC, is derived by dividing by 32 either the input (ICK) or the output (OCK) bit clock, as selected by ioc[1]. This signal provides a load signal every 32 periods of the bit clock and is used when the internal ILD is selected for the input port or the continuous internal OLD is selected for the output port. INTLOADC is synchronized to SY by each high-to-low transition of SY. The internal synchronization signal, INTSY, is derived by dividing the internal load signal by 8, 16, or 32, based on ioc[3-2].

For example, with ioc[3—0] = 5, ioc[23—20] = 6 and CKI = 49.152 MHz, the internal clock generator is configured for a 2.048 MHz bit clock, a 64 kHz load signal, and an 8 kHz synchronization signal. The signals generated by the clock generator for this case are shown in Figure 9-9a.

An internal OLD that is generated in a burst manner can also be selected. The burst internal OLD is generated (one-to-zero transition) whenever the output shift register is empty and the output buffer becomes full, or when the output buffer is full and the output shift register becomes empty. The second case provides back-to-back serial output transmissions. The burst internal OLD signal, INTLOADB, is normally high and makes a single clock cycle low pulse to indicate the start of an output. Figure 9-9b illustrates how OLD is generated with respect to the output buffer status and shift register status for the second case described above.

9.1.5.2 Clock Generator Timing Specifications

Typical uses of the clock generator are shown in Figures 9-10a—c, for ICK configured as positive edge-triggered. The timing requirements and characteristics are specified as follows:

- **Bit Clock Characteristics:** The period of the data bit clock ICK (tICKHICKH) and OCK (tOCKHOCKH), the high time of ICK (tICKHICKL) and OCK (tOCKHOCKL), and the low time of ICK (tICKLICKH) and OCK (tOCKLICKH) are shown in Figure 9-10a.
- **Load and SY Characteristics:** The delay of the ILD or OLD, and SY, when configured as internal mode from ICK (tICKHILDL, tICKHSYL) or OCK (tOCKHOLDL, tOCKHSYL) based on ioc[1] are shown in Figure 9-10b.
- **External SY Requirements for Synchronization:** When an external SY is used to synchronize the internal clock generator, a high-to-low transition on SY is required. The SY is set up high before the rising edge of ICK or OCK depending on ioc[1] (tSYHICKH, tSYHOCKH), and is held high until after the rising edge (tICKHSYX, tOCKHSYX). SY is set up low before the next rising edge (tSYLICKH, tSYLOCKH) and held low after that edge (tICKHSYX, tOCKHSYX).
- **Load Characteristic with External SY:** The internal load is synchronized on every high-to-low transition of SY. After SY goes low, ILD and OLD will be low (tSYLILDL,tSYLOLDL).

9.1.5.3 Loss of Sanity

Note: Error Condition Handling for Loss of Sanity is not supported in the DSP3210.

Loss of sanity is defined to be a high-to-low transition on SY while ioc[15], SAN, is a one. To make use of this feature, SAN should be set to one at the beginning of the program processing the frame of data, and cleared to zero before waiting for SY to be zero. If the program processing the frame takes longer that one frame interval, then the loss of sanity error is reported to the error exception handler.

9.1.5.4 Loss of Sync

Note: Error Condition Handling for Loss of Sync is not supported in the DSP3210.

Loss of sync is defined to be the occurence of an INTSY high-to-low transition (from the on-chip SIO clock generator) without a corresponding external SY high-to-low transition (from the SY pin), when SY is applied externally (ASY = 0). Note that the INTSY is synchronized (divide down counters are set to zero) by a high-to-low transition on the SY pin. For the loss of sync error to be detected, ioc[0] must be 0, and ioc[1—3] must be set so that the INTSY is the same frequency as the SY pin. If a loss of sync is detected, an error is reported to the error exception handler.

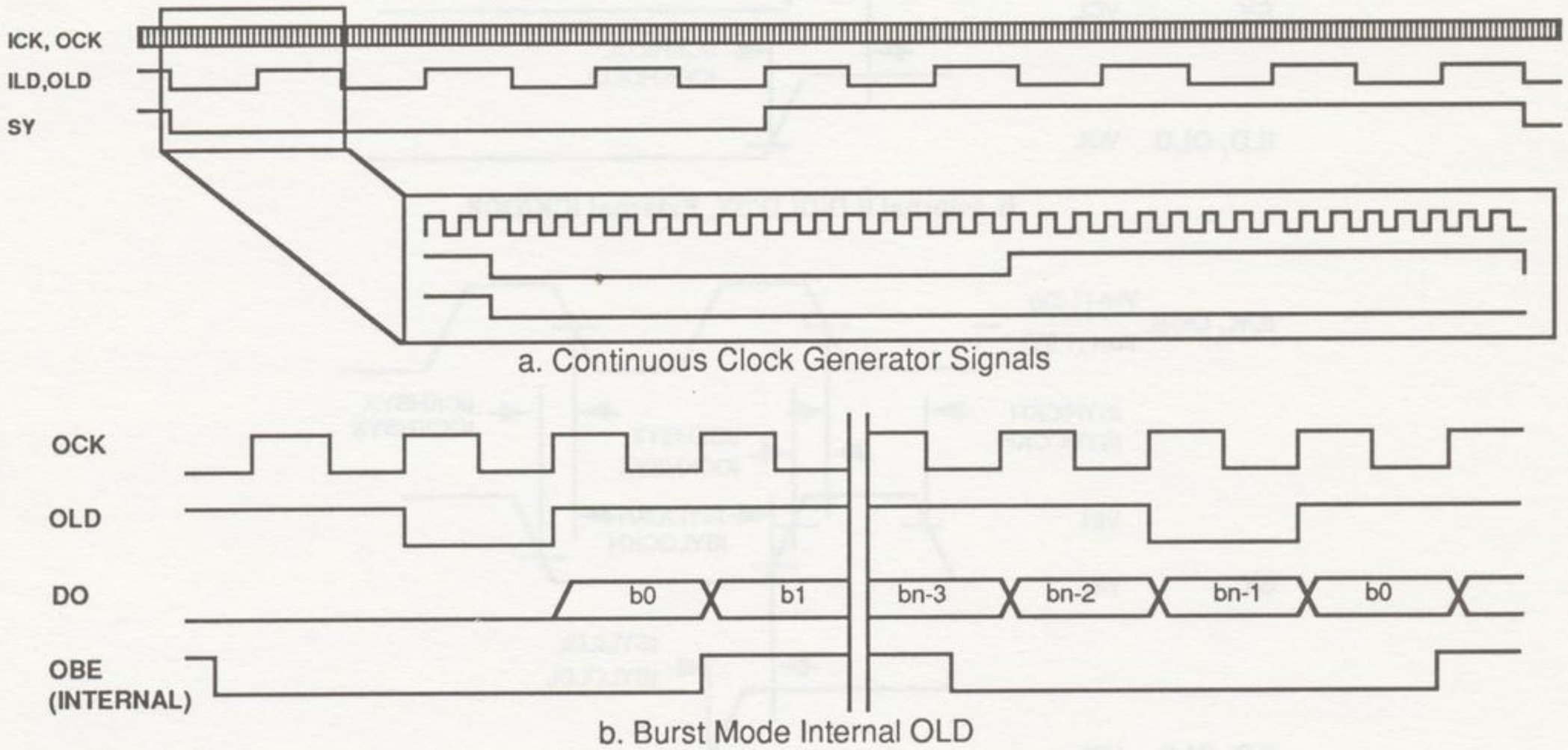
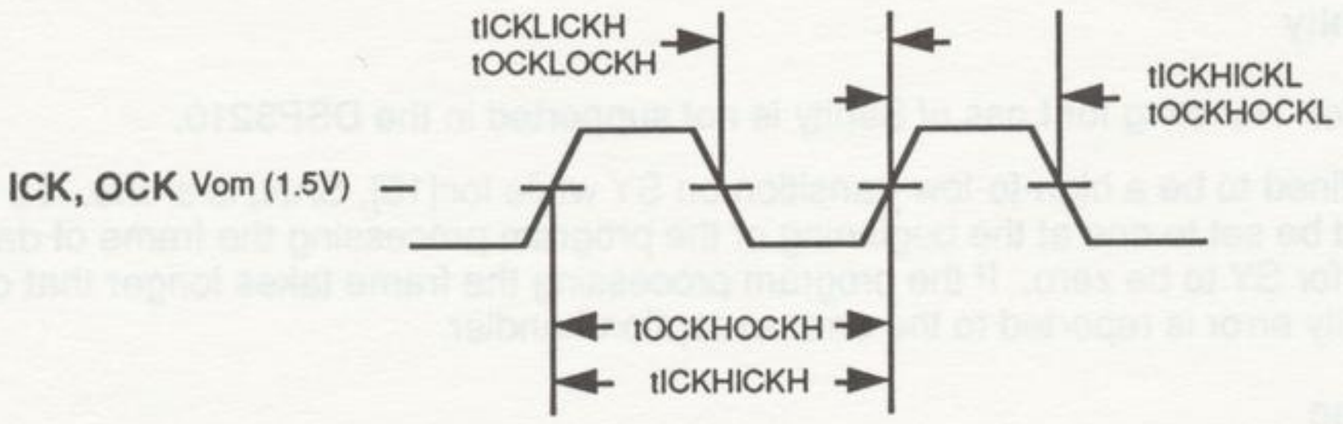
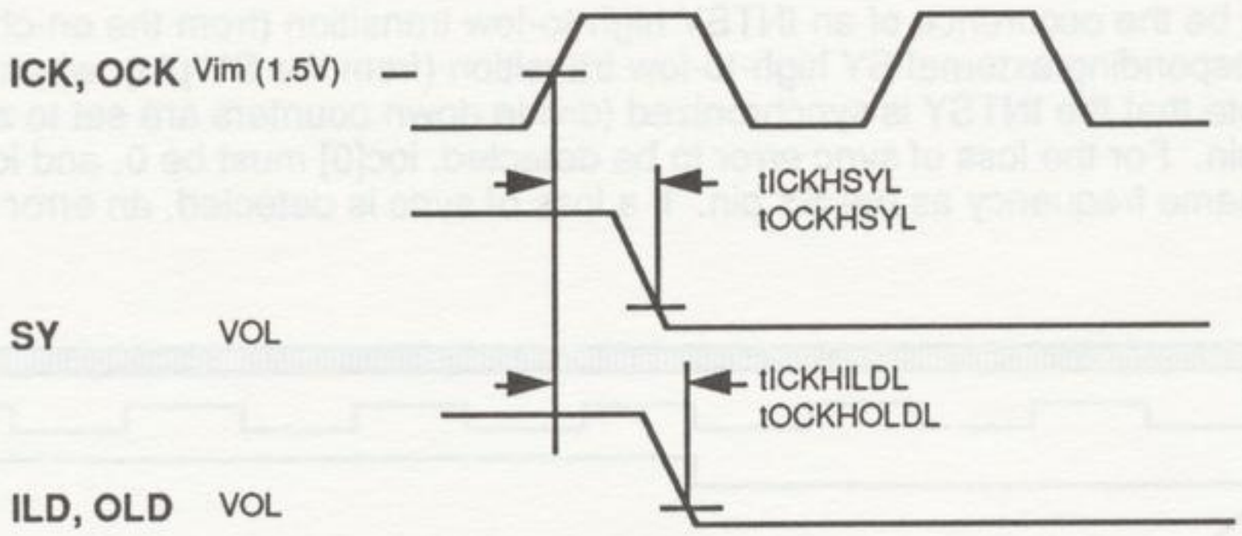


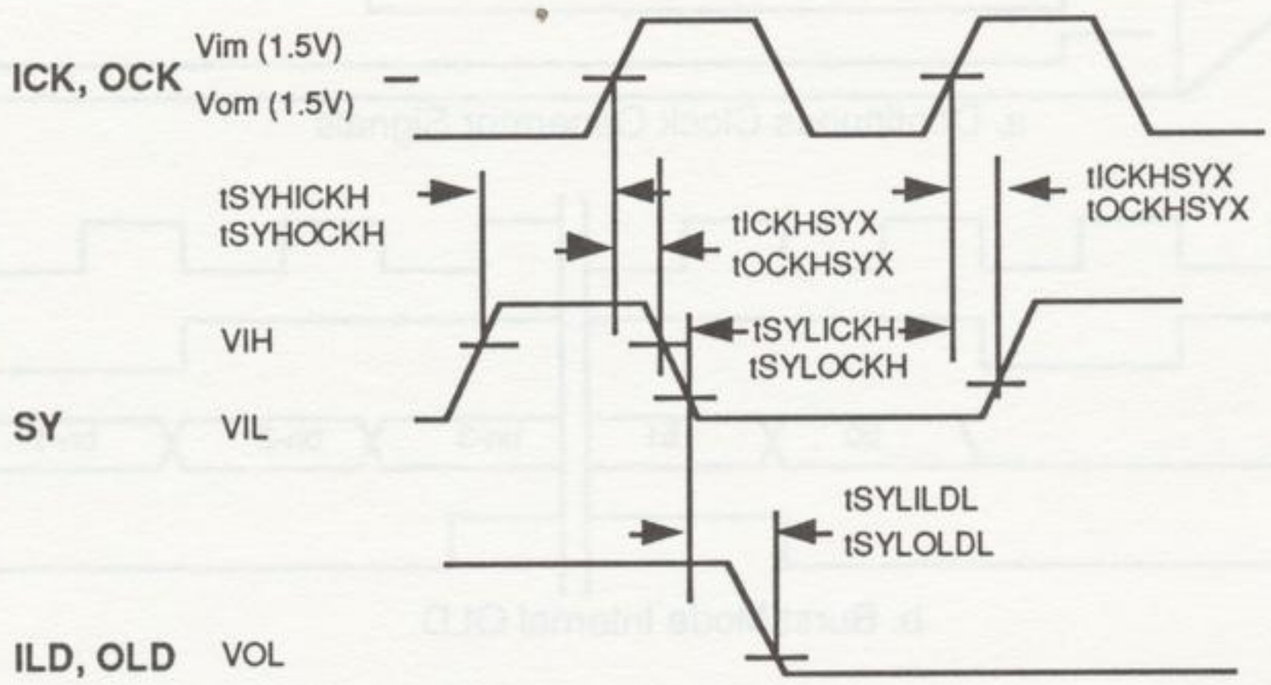
Figure 9-9 Clock Generator Signals



a. Internal ICK/OCK



b. Internal ILD/OLD/SY. External ICK/OCK



c. External SY. Internal ILD/OLD.
External or Internal ICK/OCK.

Figure 9-10 Clock Generation Timing (IIC = 0)

9.1.5.5 Frame Boundary Condition

The on-chip clock generator detects the frame boundary (fb) condition which is useful in certain TDM applications. The fb condition is defined to be the first time the output buffer is full following a high-to-low transition on SY, provided that ioc[15], SAN, is equal to zero at that time. It uniquely indicates the first channel in a TDM stream. After the SY transition is detected while SAN is zero, the fb flag will be the inverse of OBE until it is cleared to zero by setting SAN to a one. The frame boundary condition can be tested using conditional branch instructions, e.g. if(fbs)goto label (see 4.3.1 Control Arithmetic Instructions - Control Group). The frame boundary condition is usually used in conjunction with the loss of sanity error condition because toggling of the SAN bit is required for both. Figure 9-11 illustrates how the frame boundary condition is used in a TDM application.

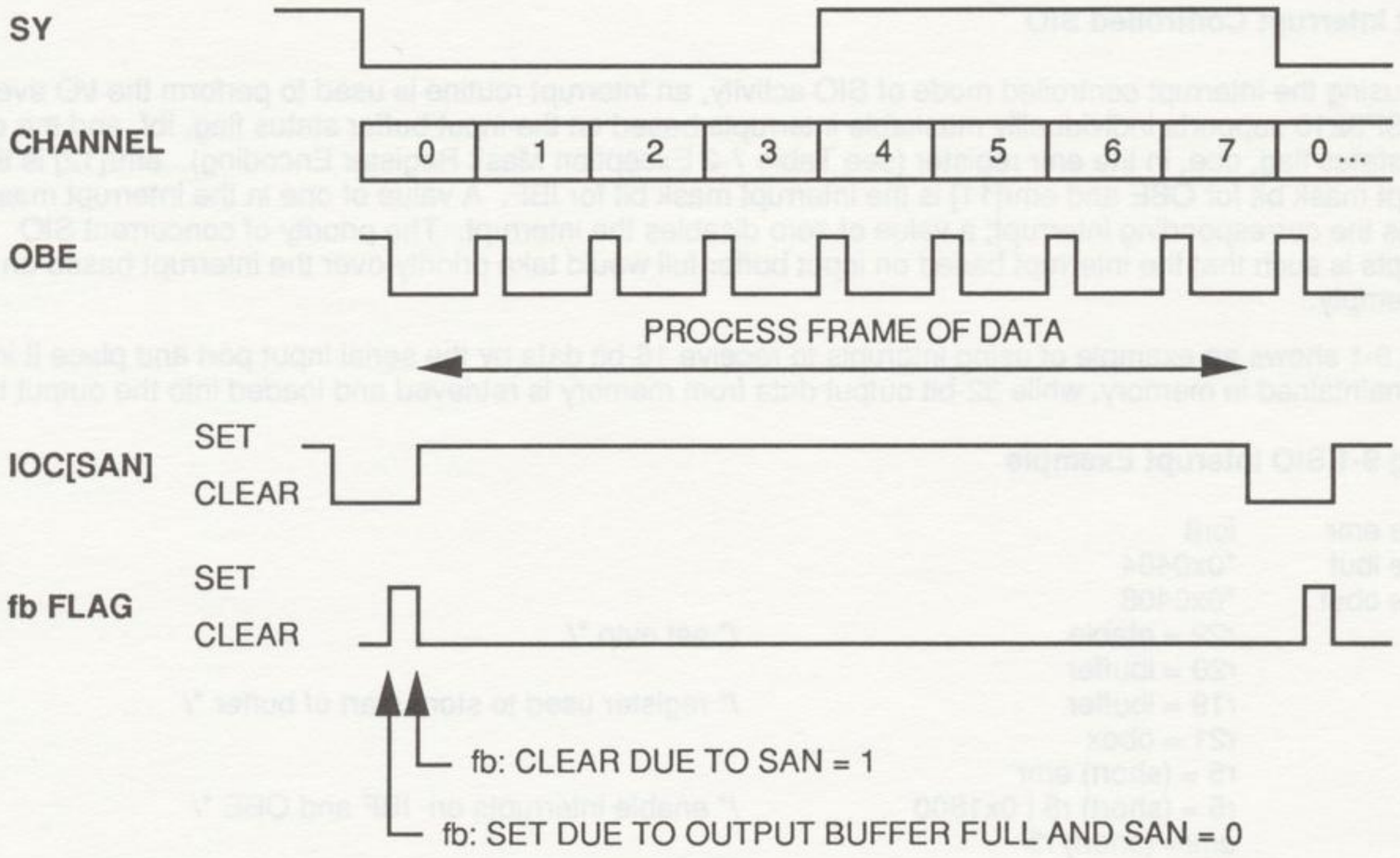


Figure 9-11 Frame Boundary Condition

9.1.6 Handling SIO Activity

This section describes three typical ways of handling SIO activity: program controlled, interrupt controlled, and DMA controlled. An example is given for each.

9.1.6.1 Program Controlled SIO

When using the program controlled mode of SIO activity, flags pertaining to the status of the input buffer, output buffer, or data stream are tested by the program. When the appropriate condition becomes true, an I/O event is initiated by the program. There are four flags available to the programmer, ibf, obe, sy, and fb. Both true and false states of these flags are directly testable by an instruction of the form:

if (IO COND) goto {rH, N, rH+N}

For more information, see 4.3.1 Control Arithmetic Instructions - Control Group.

For example, in order to process input data as it is received by the serial input port, the program must wait until the input buffer is full. This is accomplished by the following instructions:

```
waitibf:    if(ibe) pcgoto waitibf      /* conditional branch using pc relative addressing */
            nop                         /* latent instruction */
```

9.1.6.2 Interrupt Controlled SIO

When using the interrupt controlled mode of SIO activity, an interrupt routine is used to perform the I/O event. The DSP3210 supports individuallly maskable interrupts based on the input buffer status flag, ibf, and the output buffer status flag, obe, in the emr register (see Table 7-2 Exception Mask Register Encoding). emr[12] is the interrupt mask bit for OBE and emr[11] is the interrupt mask bit for IBF. A value of one in the interrupt mask bit enables the corresponding interrupt; a value of zero disables the interrupt. The priority of concurrent SIO interrupts is such that the interrupt based on input buffer full would take priority over the interrupt based on output buffer empty.

Listing 9-1 shows an example of using interupts to receive 16-bit data by the serial input port and place it in a FIFO maintained in memory, while 32-bit output data from memory is retrieved and loaded into the output buffer.

Listing 9-1 SIO Interupt Example

```
#define emr     ior8
#define ibuf    *0x0404
#define obuf    *0x0408
                r22 = etable                    /* set evtp */
                r20 = ibuffer
                r19 = ibuffer                   /* register used to store start of buffer */
                r21 = obox
                r5 = (short) emr
                r5 = (short) r5 | 0x1800        /* enable interrupts on  IBF and OBE */
                emr= (short) r5
main:           .                               /* program body */
                .
                .
                pcgoto main
ibuffer:        N*int                           /* FIFO storage in memory  */
obox:           float
etable:         22*nop                          /* First eleven entries of evt */
                pcgoto ibfisr                   /* branch to IBF interrupt service routine */
                nop
                pcgoto obeisr                   /* branch to IBF interrupt service routine */
                nop
                8*nop                           /* Remainder of evt */
ibfisr:         r1= (short) ibuf
                nop
                *r20++ = (short) r1             /* write ibuf to memory */
                r20 - r21                       /* compare r20 to end of buffer  */
                if (mi) r20 = r19               /* reload starting address if at end */
                ireturn
obeisr:         r1=*r21
                nop
                obuf=r1                         /* load obuf from r1 */
                ireturn
```

9.1.6.3 DMA Controlled SIO

The DSP3210 includes a DMA option for transferring data between the ibuf, obuf, and memory without program intervention. The serial DMA option is particularly useful for multichannel applications in which input and output buffers for an entire frame of samples are established in memory.

The DSP3210 contains a DMA controller block to perform the necessary address generation and control for the DMA process (see Section 9.2 DMA Controller). The DMA controller consists of two channels, one for input DMA and one for output DMA. The registers to configure and enable DMA activity are MMIO registers and are described fully in Section 9.2. All serial I/O DMA activity is based on either the IBF flag or the OBE flag. Based on one or both of these flags, an input DMA will occur, an output DMA will occur, or both an input and an output DMA will occur. The last type will perform both operations, input DMA and then output DMA, in one DMA cycle, where a DMA cycle is equal to one instruction cycle.

To buffer a frame of serial input data in DSP3210 memory, the following registers need to be setup:

icnt	transfer count for input DMA
idp	input dma pointer
dmac	bits 4-7 define the mode of input DMA

For a buffer of 128, 16-bit samples, the icnt register should be configured for 0x00fe0000 (one less than the number of transfers desired times the data size in bytes). The idp should be set for the base address of the buffer which in this case must be aligned for 16-bit data (bit 0 must be 0). After the icnt and idp registers have been initialized, the dmac is loaded with the desired configuration. Program Listing 9-2 shows the code necessary to initialize the input channel of the DMA controller and enable input DMA. An interrupt can be taken when the input DMA transfer is complete by unmasking the IFRM bit of the emr register and supplying an appropriate interrupt routine.

Listing 9-2 Input DMA Example

```
#define icnt    *0x428
#define idp     *0x420
#define dmac    *0x430

        r1= 0xfe        /* icnt for 128, 16-bit samples */
        icnt=r1
        r1=ibuffer      /* starting address of input buffer */
        idp=r1
        r1=0x30         /* enable input DMA, and stop when transfer count is reached */
        dmac=r1

ibuffer: 128*int        /* input DMA buffer */
```

9.2 DMA Controller (DMAC)

The DMA controller is a two channel Direct Memory Access (DMA) control unit. It is used in conjunction with the Serial I/O to buffer input and output data streams in DSP3210 memory, internal or external. The DMA transfers occur between memory and the SIO ports without processor intervention, i.e., using cycle-stealing where the DSP3210 waits during a DMA transfer (see Section 7.3 Direct Memory Access).

9.2.1 DMAC Programming Model

The register programming model for the DMA controller consists of five MMIO registers. The idp and the odp are the input DMA and the output DMA pointer registers, respectively. Each is 32-bits and therefore can point to any location in the DSP3210 address space. When read, the idp and odp return the value of the current pointer register. The icnt and ocnt registers define the transfer count for input and output DMA frames, respectively. A DMA frame is considered to be the number of DMA transfers to be performed. The transfer count is the number of bytes to be transferred. **The transfer count is specified in bytes and is set to one transfer less than the frame size;** e.g., for an output frame size of 64 and a transfer size of 16-bits the ocnt register is set for 126 or (64 – 1) * 2. When read, the icnt and ocnt return the value of the current transfer count register. The dmac register is used to configure the operating modes of both DMA controllers. The dmac register is set to zero on reset. The encoding and functional definition of each bit field of the dmac register are described in Table 9-3. The address is given for both big and little endian byte orderings. The byte order is specified by pcw[8]. The register programming model for the DMAC is pictured below:

Register Name	Address Big Endian	Address LIttle Endian	Register Bits 31 … 16 15 … 8 7 … 0	
idp	0x0420	0x0420		R/W
odp	0x0429	0x0424		R/W
icnt	0x042a	0x0428		R/W
ocnt	0x042e	0x042C		R/W
dmac	0x0433	0x0430		R/W

9.2.2 DMAC Functional Description

Each DMA controller consists of a base address register, a transfer count register, an adder, a counter, and a comparator (see Figure 9-12). A DMA channel can be configured to operate in one of two primary modes: stop on transfer complete mode or reload mode. The stop on transfer complete mode is used to transfer a single frame of data, and the reload mode is normally used when transferring continuous frames of data. In the reload mode, the base address register and the transfer count register operate in a double buffered manner. The register currently being used is called the **current** register, and the register to be used next is called the **next** register.

In the stop on transfer complete mode (ORS = 0 or IRS = 0), when the processor loads the idp, odp, icnt, or ocnt registers, both the **next** register and the **current** register are written, and the transfer counter is set to zero. The address of the current location being pointed to is generated by adding the counter value to the current DMA pointer register. The comparator is used to compare the counter output to the current transfer count. When the counter equals the current transfer count the enable/disable bit (ODE or IDE) is set to zero, and the counter is set to zero.

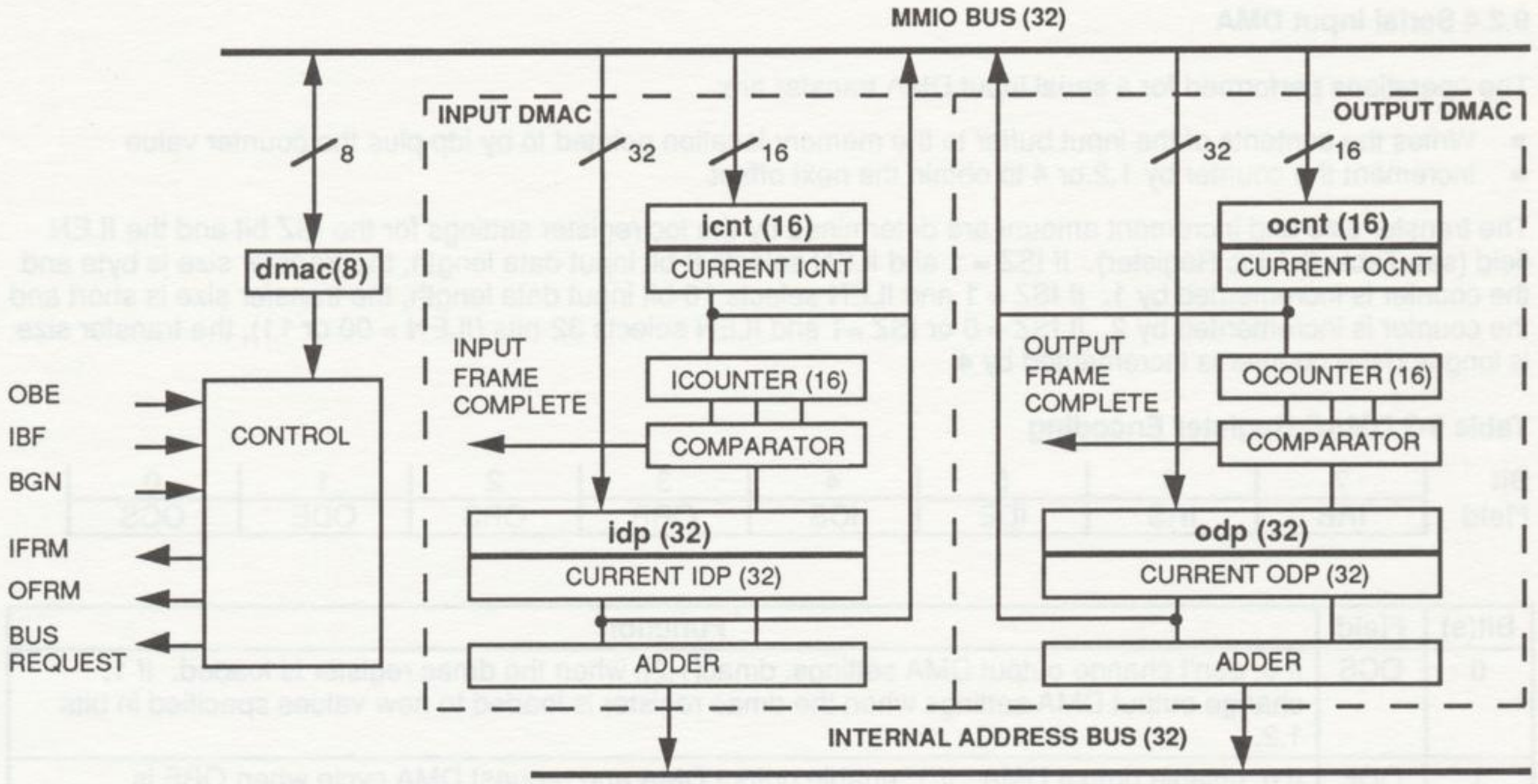


Figure 9-12 DMAC Block Diagram

In the reload mode (ORS = 1 or IRS = 1), when the processor loads the idp, odp, icnt, or ocnt registers, only the **next** register is affected. When the counter equals the current transfer count, the current pointer register and current transfer count registers are loaded from the next pointer register and next transfer count registers, respectively, and the transfer counter is set to zero. The enable/disable bit is unaffected. When the first frame is being set up, the dmac register should be configured such that DMA is disabled and the channel is in the stop on transfer complete mode.

A mode is provided so that DMA cycles are requested only when the DSP3210 has ownership of its bus interface. This is useful when the DSP3210 is buffering data to or from a shared external memory, because it guarantees that the DMA cycle will not incur the overhead of waiting to gain ownership of the bus. This mode is enabled with the ORB for output DMA and IRB for input DMA. It operates by requesting the bus interface when a DMA operation has been requested, but not performing the DMA cycle until the bus has been granted.

Note: When a DMA controller is enabled, the mode of operation, as specified by the dmac register, and the size of the transfer, as specified by the ioc register, should not be changed.

9.2.3 Serial Output DMA

The operations performed for a serial output DMA transfer are:

- write the contents of the memory location pointed to by odp plus the counter value to the output buffer
- increment the counter by 1,2,or 4 to obtain the next offset.

The transfer size and increment amount are determined by the ioc register settings for the OSZ bit and the OLEN field (see Table 9-1 ioc Register). If OSZ = 1 and OLEN selects 8-bit input data length, the transfer size is byte and the counter is incremented by one. If OSZ=1 and OLEN selects 16-bit input data length, the transfer size is short and the counter is incremented by 2. If OSZ = 0 or OSZ = 1 and OLEN selects 32-bits (OLEN = 011 or 100), the transfer size is long and the counter is incremented by 4.

9.2.4 Serial Input DMA

The operations performed for a serial input DMA transfer are:

- Writes the contents of the input buffer to the memory location pointed to by idp plus the counter value
- Increment the counter by 1,2,or 4 to obtain the next offset

The transfer size and increment amount are determined by the ioc register settings for the ISZ bit and the ILEN field (see Table 9-1 ioc Register). If ISZ = 1 and ILEN selects 8-bit input data length, the transfer size is byte and the counter is incremented by 1. If ISZ = 1 and ILEN selects 16-bit input data length, the transfer size is short and the counter is incremented by 2. If ISZ = 0 or ISZ =1 and ILEN selects 32-bits (ILEN = 00 or 11), the transfer size is long and the counter is incremented by 4.

Table 9-3 DMAC Register Encoding

Bit	7	6	5	4	3	2	1	0
Field	IRB	IRS	IDE	ICS	ORB	ORS	ODE	OCS

Bit(s)	Field	Function
0	OCS	If 0, don't change output DMA settings, dmac[1,2], when the dmac register is loaded. If 1, change output DMA settings when the dmac register is loaded to new values specified in bits 1,2.
1	ODE	If 0, disable output DMA. If 1, enable output DMA and request DMA cycle when OBE is asserted, obuf is empty.
2	ORS	If 0, stop output DMA when the number of DMA transfers (OCOUNTER) reaches the number specified in OCNT. If 1, reload the current output DMA pointer and the current output transfer count registers, ODP and OCNT, respectively, when the number of DMA transfers (OCOUNTER) reaches the number specified in OCNT and keep output DMA enabled.
3	ORB	If 0, request output DMA cycles based on OBE and dmac[1]. If 1, request output DMA cycles based on OBE, dmac[1], **and** if the DSP3210 has ownership of the bus interface. If the DSP3210 is not the current bus master, the BRN is asserted. When the bus is granted, the DMA cycle is requested.
4	ICS	If 0, don't change input DMA settings, dmac[4,5], when the dmac register is loaded. If 1, change input DMA settings when the dmac register is loaded to new values specified in bits 4,5.
5	IDE	If 0, disable input DMA. If 1, enable input DMA and request DMA cycle when IBF is aserted, ibuf is full.
6	IRS	If 0, stop input DMA when the number of DMA transfers (ICOUNTER) reaches the number specified in ICNT. If 1, reload the current input DMA pointer and the current input transfer count registers, IDP and ICNT, respectively, when the number of DMA transfers (ICOUNTER) reaches the number specified in ICNT and keep output DMA enabled.
7	IRB	If 0, request input DMA cycles based on IBF and dmac[4]. If 1, request DMA cycles based on IBF, dmac[4], **and** if the DSP3210 has ownership of the bus interface. If the DSP3210 is not the current bus master, the BRN is asserted. When the bus is granted, the DMA cycle is requested.

9.3 Timer

The timer is a programmable 32-bit counter and associated control register and logic that can be used for interval timing, rate generation, event counting, or waveform generation. The input to the timer can be derived from the DSP3210 clock or come from an external source. The output of the timer can generate a maskable interrupt or be selected as an output of the chip to drive external hardware. The count-down timer can be configured to count to zero once, or to count continuously by automatically reloading the counter with its initial value when it reaches zero.

9.3.1 Timer Programming Model

Access to the timer is provided by reading and writing the **tcon** and **timer** MMIO registers. When the **timer** is written, the initial count register and counter are loaded. When the timer is read, the current counter value is read. The count value may be read or changed at any time during operation. The tcon register is used to configure and control the timer. Table 9-4 describes the function of each bit of the tcon register. On reset, all bits of the tcon register are set to zero and the timer is set to all 1s. The address for the timer registers is given for big and little endian byte ordering. The byte ordering is specified by pcw[8]. The register programming model for the timer is:

Register Name	Address Big Endian	Address LIttle Endian	Register Bits	
			31 ... 7 ... 0	
tcon	0x0413	0x0410	[ ]	R/W
timer	0x0414	0x0414	[ ]	R/W

9.3.2 Timer Functional Description

The timer consists of an 8-bit control register (tcon), the initial count register, the counter, and associated control logic (source selection, counter control, and output control), see Figure 9-13. The source selection logic is used to select the clock source for the counter and is based on the value of the tcon register. The source can be a divide down of the DSP system clock, CKI, or the source can be the external signal applied to the pin BIO0, Bit I/O 0. Divide-down rates for CKI of 2, and 4 can be selected. The true or complement of the external source, BIO0, can be selected. The maximum frequency of an external source is CKI/2. The source selection logic detects zero-to-one transitions of the selected clock source.

The counter control logic controls the loading and decrementing of the counter. The counter is loaded when the timer is written, and when the count reaches zero if tcon[1] = 1 (automatic reload of counter). If the counter is loaded when it is enabled, the loading of the counter takes preference over decrementing. The counter is decremented by one for each zero-to-one transition of the clock source detected by the source selection logic. At any time, the counter can be stopped and the count held by setting tcon[0] to a zero. Setting tcon[0] to a one at this time will continue the count-down operation from the current count.

The output control logic controls the type of output signal generated by the counter. Whenever the count reaches zero, an interrupt request is generated, the state of a toggle flip-flop changes state, and a pulse, one CKI period wide, is generated. In order for the DSP to respond to the interrupt, the timer interrupt bit in the emr register, emr[9], must be set to one (unmasked) (see Table 7-2 emr Register Encoding). Depending on the setting of tcon[2], either the output of the toggle flip-flop or the pulse will be available to be observed external to the DSP via pin BIO1. For this, pin BIO1 has to be configured as an output and tcon[4] has to be set to one. If the pulse mode is selected (tcon[2] = 0), the pulse can be active high or active low, depending on tcon[3]. The active high signal is normally low, and when the count reaches zero it will pulse **high** for one clock period. The active low signal is normally high, and when the count reaches zero it will pulse **low** for one clock period.

Use of the timer usually starts with loading the initial count register and the counter. This is done by loading the **timer** register. After loading the counter, the count-down operation is enabled by setting tcon[0] to a one. When the count reaches zero and if tcon[1] is set to a one, the counter is reloaded automatically from the initial count register, without the need for program intervention. The period of the count is N+1, where N is the value loaded in the counter. Alternatively, when the count reaches zero and if tcon[1] is set to a zero, the counter stops. In this case, the counting interval is N, where N is the value loaded into the counter.

By setting the counter to zero and configuring it to reload continuously, it is possible to generate a clock, observable externally at the BIO1 pin, which has either a CKI/2 or a CKI/4 rate. For example, a setting of TCON to 0x13 will result in an external clock with a rate of CKI/2 and 50% duty cycle; a setting of TCON to 0x17 generates an external clock with a rate of CKI/4 and 50% duty cycle; a setting of TCON to 0x33 generates an external clock with a rate of CKI/4 and 75% duty cycle (3 CKI periods high, 1 CKI period low); and a setting of TCON to 0x3b will give an external clock with a rate of CKI/4 and 25% duty cycle (1 CKI period high, 3 CKI periods low).

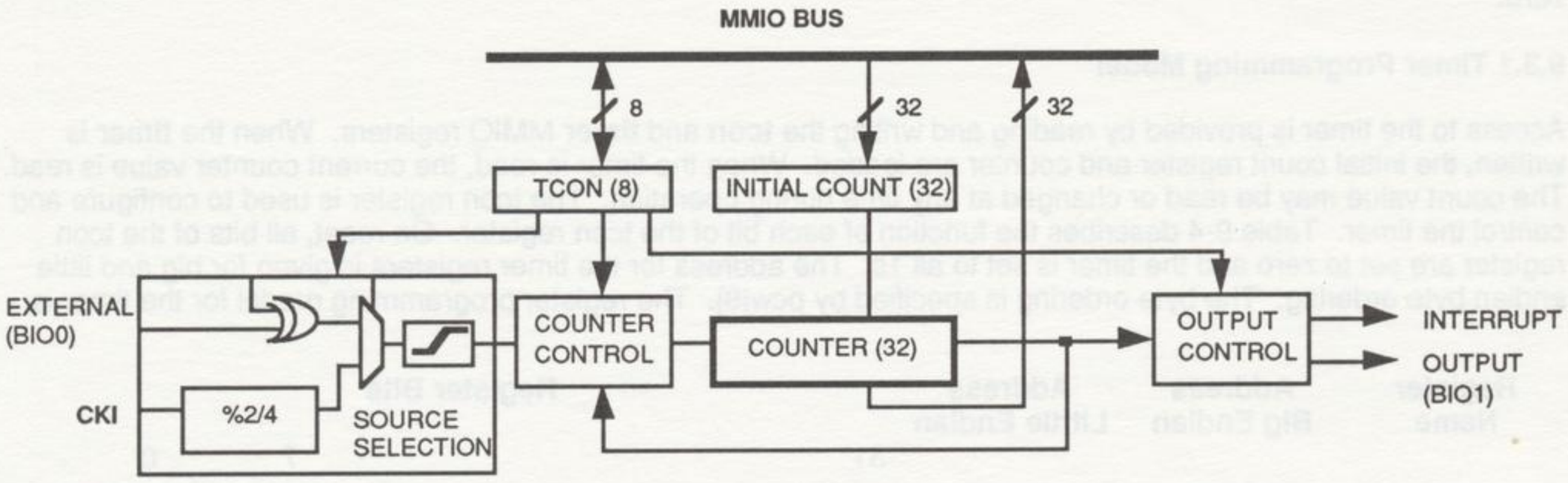


Figure 9-13 Timer Block Diagram

Table 9-4 TCON Register Encoding

Bit	7 6 5	4	3	2	1	0
Field	SRC	OUT	H/LN	T/PN	R/SN	E/HN

Bit(s)	Field	Function
0	E/HN	Hold/enable count-down operation of counter. 0 - Hold the count. 1 - Enable the count-down operation of counter.
1	R/SN	Select between continuous and one-time operation. 0 - Stop counting when the count reaches zero. 1 - Automatic reload of the counter from the initial count register when the count reaches zero.
2	T/PN	Select pulse or toggle operation of the output when the count reaches zero. 0 - For the external output, a pulse is generated when the count reaches zero. 1 - For the external output, the output is the output of a toggle flip-flop which is toggled when the count reaches zero.
3	H/LN	When tcon[2] is set to zero, pulse, select low active or high active pulse. 0 - When tcon[2] is set to zero, the pulse is one CKI clock period wide and is low active (1->0->1). 1 - When tcon[2] is set to zero, the pulse is one CKI clock period wide and is active high (0->1->0).
4	OUT	If the BIO1 pin is configured as an output, 0 - The BIO1 pin has the value specified by the BIO1 field in the BIO register. 1 - The BIO1 pin is selected to be the timer output.
5-7	SRC	Selects the clock source for the timer. tcon 7,6,5 000 Counter clock is CKI/2 001 Counter clock is CKI/4 010 Reserved 011 Reserved 100 Counter clock is from external source(pin BIO0) - rising edge 101 Counter clock is from external source (pin BIO0) - falling edge 11x Reserved

9.3.3 Timer Timing Specifications

When BIO0 or BIO1 are selected to be used as the timer clock source or the timer output, respectively, the following timing specifications apply. BIO0 is an asynchronous input and is synchronized to the DSP3210 by an internal version of CKI. If an input makes a transition during the sample window, the level recognized by the processor is not predictable; however, the processor always resolves the latched level to either a logic one or zero before using it. Figure 9-14 shows the sampling window for BIO0 and the timing relationship between BIO1 and CKI. BIO1 changes on the falling edge of CKI. The specifications for BIO0 when it is asynchronous to the DSP3210 are that the period (tBIO0HBIO0H) must be greater than twice the CKI period, and the high or low time (tBIO0HBIO0L and tBIO0LBIO0H) must be greater than one CKI period.

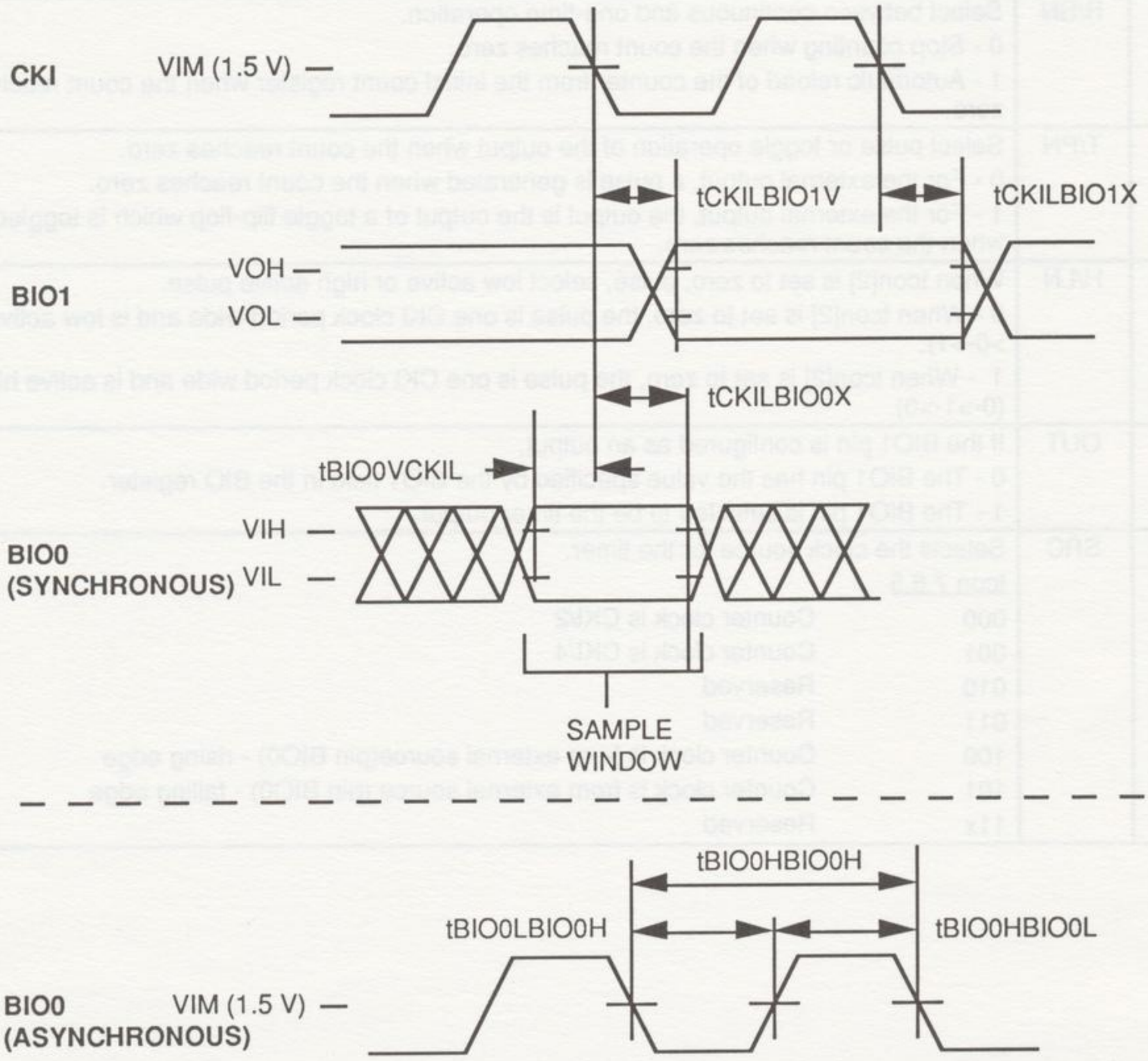


Figure 9-14 Timer Timing Specifications

9.4 Bit I/O (BIO)

The BIO is a general-purpose 8-bit input/output port. It includes features that make it suitable for board-level status signal generation and control signal testing by the DSP3210. The BIO interface consists of 8 I/O lines, any of which can be independently configured as an input or an output. Outputs can be written with a 1 or a 0, toggled, or left unchanged. Inputs can be directly read and loaded into a CAU register and then tested.

9.4.1 BIO Programming Model

Access to the BIO is provided by reading and writing the **bioc** and **bio** MMIO registers using CA Data Move Instructions (e.g. *0x0418 = (byte) r1, r1 = (short) *0x041C, ...). The bioc register is used to configure each bit of the BIO as an input or an output. Table 9-5 describes the function of each bit of the bioc register. The bio register is used to alter the state of bits configured as outputs or to determine the logical state of BIO inputs. Table 9-6 describes the function of each bit of the bio register. On reset, all bits of the bioc register and output register bits of the bio register are set to zero. The register programming model for the BIO is as follows:

Register Name	Address Big Endian	Address LIttle Endian	Register Bits 31 … 16 15 … 8 7 … 0	
bioc	0x041b	0x0418		R/W
bio	0x041e	0x041C		R/W

Table 9-5 bioc Register Encoding

Bit	7	6	5	4	3	2	1	0
Field	B7	B6	B5	B4	B3	B2	B1	B0

Each bit in the bioc register corresponds to one of the pins in the BIO port, B0—BIO0, B1—BIO1, ..., B7—BIO7. A value of 0 in the bit indicates that the associated pin is configured as an input; a value of 1 indicates that the pin is configured as an output. On reset all bits are set to zero (input).

Table 9-6 BIO Register Encodings (Writing and Reading)

Bit	15 14	13 12	11 10	9 8	7 6	5 4	3 2	0 1
Field	BF7	BF6	BF5	BF4	BF3	BF2	BF1	BF0

Each pair of bits (field) in the bio register corresponds to one of the bio pins in the BIO port, BF0-BIO0, BF1—BIO1, ..., BF7—BIO7. When the bio register is written, the value of each output register bit is changed according to the following encoding of the bit field:

BF bit pair: bit 1,0

00	Keep the previous value in the bio output register bit
01	Set the bio output register bit to zero
10	Set the bio output register bit to one
11	Complement the value in the bio output register bit

The transfer function is described below, where Qt is the previous value of an output register bit and Qt+1 is the new value:

Qt	Bit1	Bit0	Qt+1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

When the bio register is read, the value of both the corresponding bio pin and the bio output register bit are read according to the following encoding of the bit field:

bit 0	Bio pin value
bit 1	Value of the bio output register bit

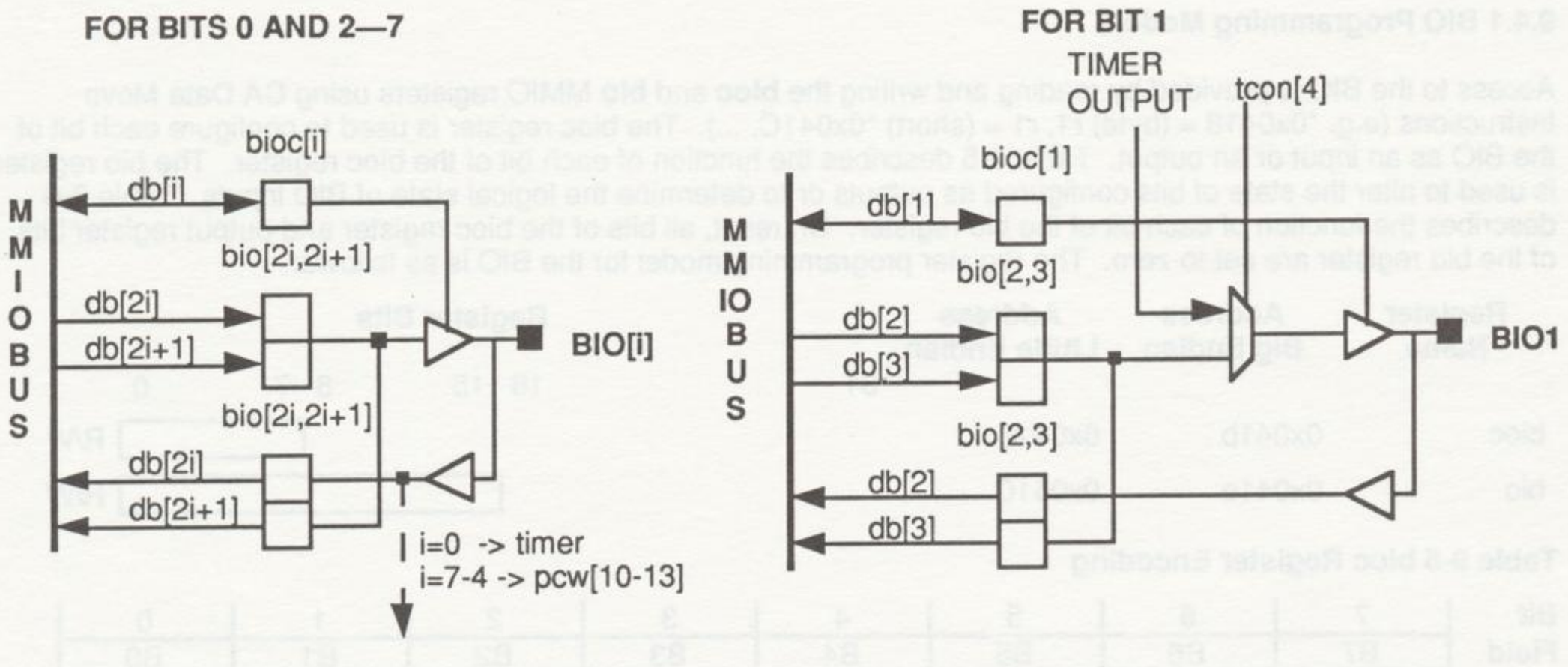


Figure 9-15 Bit I/O Bit Slice Diagrams

9.4.2 BIO Functional Description

Figure 9-15 shows bit slices for the two types of bits in the BIO. The structure for Bit 0 and bits 2—7 are the same, except that the BIO0 pin can be used as the timer source and BIO7—4 can be used to input the reset values of pcw[10—13]. Bit 1 has the additional capability of serving as the timer output, therefore an additional multiplexer controlled by tcon[4] is needed in this bit slice.

Each bit of the BIO port can be configured independently as either an input or an output by writing the desired value in the bioc (bit I/O configuration) register (see Table 9-5). If the pin is configured as an output, values written to the bio register are immediately reflected to the pin. The value of each output pin can be changed by writing the bio register with an appropriate pattern as described in Table 9-6. Each bio output register bit can be set to a 1 or a 0, toggled, or left unchanged. When the bio register is read, both the logic value of the pin and the value of the output register bit (the value that would be driven to the pin if the pin were configured as an output) are read.

9.4.3 BIO Timing Specifications

When configured as inputs by the bioc register, BIO pins are asynchronous inputs that are synchronized to the DSP3210 by an internal version of CKI. If an input makes a transition during the sample window, the level recognized by the processor is not predictable; however, the processor always resolves the latched level to either a logic one or zero before using it. Figure 9-16 shows the sampling window for BIO inputs. When configured as outputs, BIO pins change with respect to the falling edge of CKI. The timing relationship between the BIO outputs and CKI is also shown in Figure 9-16.

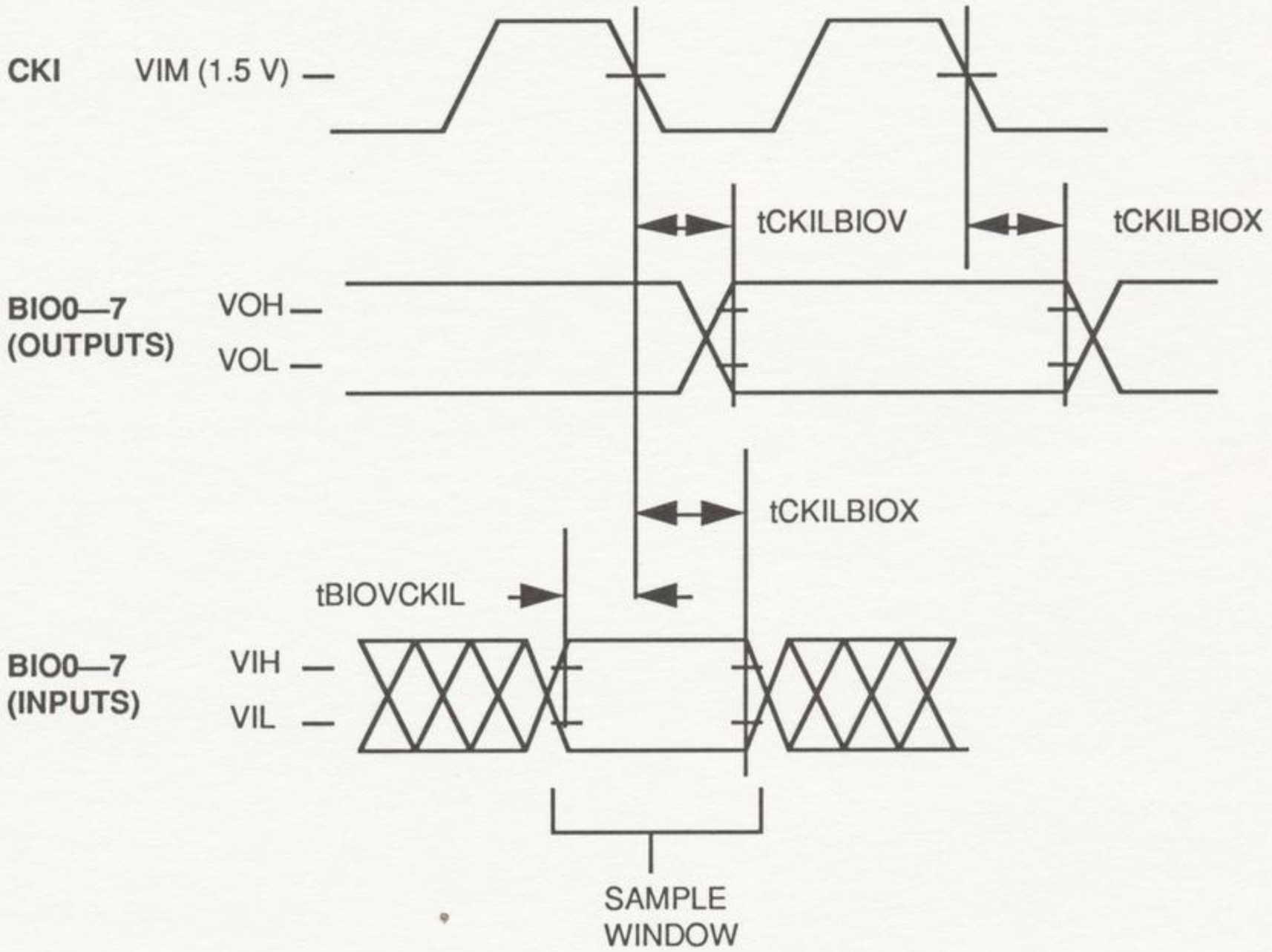


Figure 9-16 BIO Timing Specifications

Chapter 10

Instruction and Register Encodings

CHAPTER 10. INSTRUCTION AND REGISTER ENCODINGS

10. INSTRUCTION AND REGISTER ENCODINGS 10-1
10.1 Instruction Set Encodings 10-1
Table 10-1 Instruction Encodings 10-1
Table 10-2 CA Instruction Field Encodings 10-3
Table 10-3 DA Instruction Field Encodings 10-5
10.2 Register Encodings 10-6
Table 10-4 ps Register Encoding 10-6
Table 10-5 dauc Register Encoding 10-6
Table 10-6 ctr Register Encoding 10-7
Table 10-7 emr Register Encoding 10-7
Table 10-8 pcw Register Encoding 10-7
Table 10-9 ioc Register Encoding 10-9
Table 10-10 dmac Register Encoding 10-10
Table 10-11 tcon Register Encoding 10-11
Table 10-12 bioc Register Encoding 10-12
Table 10-13 bio Register Encoding 10-12

10. INSTRUCTION AND REGISTER ENCODINGS

10.1 Instruction Set Encodings

Tables 10-1 to 10-3 illustrate the bit-level encoding of the DSP3210 instruction set.

Table 10-1 Instruction Encodings

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0a & 1a. Reserved	0	0	0	0	0																											
0b & 1b. Cond Branch	1	0	0	0	0	C											rB															
																	N															
2a. Reserved	0	0	0	0	1	0																										
2b. Reserved	1	0	0	0	1	0																										
3a. Loop Counter	0	0	0	0	1	1	rM										rB															
																	N															
3b. Do Instr. (Immed)	1	0	0	0	1	1	0	B	M						K								L									
3c. Do Instr. (Reg)	1	0	0	0	1	1	1	B	M						K														rM			
4a. Call Reg with 16-bit N	0	0	0	1	0	0	rM										rB															
																	N															
4b. Shift and Or	1	0	0	1	0	0	rD										rS															
																	N															
5a. 16-bit 3 Opnd Add	0	0	0	1	0	1	rD										rS3															
																	N															
5b. 32-bit 3 Opnd Add	1	0	0	1	0	1	rD										rS3															
																	N															
6a. 16-bit ALU Reg Src	0	0	0	1	1	0	0	F							rD						rS1					C					rS2	
6b. 32-bit ALU Reg Src	1	0	0	1	1	0	0	F							rD						rS1					C					rS2	
6c. 16-bit ALU Immed	0	0	0	1	1	0	1	F							rD						N											

Table 10-1 Instruction Encodings (cont.)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
6d. 32-bit ALU Immed	1	0	0	1	1	0	1	F				rD					N															
7a. MOVE - DA	0	0	0	1	1	1	0	T	W			rH					L															
7b. MOVE - I/O	1	0	0	1	1	1	0	T	W			rH										1							iorH			
7c. MOVE - Pnt Inc	1	0	0	1	1	1	0	T	W			rH					rP					0							rI			
7d. MOVE - Mem - I/O	1	0	0	1	1	1	1	T	W			iorH					rP					0							rI			
8a. Uncond Branch	1	0	1	Ne								rB					N															
8b. 24-Bit Immed Load	1	1	0	Ne								rD					N															
8c. Call 24-Bit Immed	1	1	1	Ne								rM					N															
1. DAU - M/A	0	0	1	M			F	S	N	X							Y											Z			
2. DAU - M/A	0	1	0	M			F	S	N	X							Y											Z			
3. DAU - M/A	0	1	1	M			F	S	N	X							Y											Z			
4. DAU - M/A	0	0	1	1	1	0		F	S	N	X						Y											Z			
5. DAU - Spec. Func.	0	1	1	1	1	G			N	0	0	0	0	0	0	0	Y											Z			

Table 10-2 CA Instruction Field Encodings

CA - W Field					
	Bytes				
W	3	2	1	0	Operation
000	0	0	0	D	unsigned char
001	S	S	S	D	signed char
010	0	0	D	D	unsigned short
011	S	S	D	D	signed short
100	0	0	D	0	unsigned high char
101	-	-	-	-	reserved
110	-	-	-	-	reserved
111	D	D	D	D	long

CA - rB, rM, rH, rD, rS, rS1, rS2, rS3,rP, rI			
CODE	CAU Register	CODE	CAU Register
00000	0 (r0)	10000	Reserved
00001	r1	10001	r15
00010	r2	10010	r16
00011	r3	10011	r17
00100	r4	10100	r18
00101	r5	10101	r19
00110	r6	10110	-n
00111	r7	10111	+n
01000	r8	11000	r20
01001	r9	11001	r21
01010	r10	11010	r22
01011	r11	11011	Reserved
01100	r12	11100	Reserved
01101	r13	11101	Reserved
01110	r14	11110	pcsh
01111	pc	11111	Reserved

CA - R Field			
R	I/O Register	R	I/O Register
00000	ps	01010	spc
00001	Reserved	01011	Reserved
0001x	Reserved	01100	pcw
001xx	Reserved	01101	Reserved
01000	emr	01110	dauc
01001	Reserved	01111	ctr
		1xxxx	Reserved

CA - F Field	
F	ALU/BS Function
0000	+ addition
0001	<<n shift left by n
0010	- subtract {N} - rD
0011	# carry reverse add
0100	- subtract rD - {N, rS}
0101	reserved
0110	&~ and with complement
0111	- compare (no store)
1000	^ exclusive or
1001	>>>1 rotate right by 1
1010	| bitwise OR
1011	<<<1 rotate left by 1
1100	>>n logical shift right by n
1101	$>>n arithmetic shift right by n
1110	& bitwise AND
1111	& bitwise AND (no store)

CA - T Field	
T	Operation
0	Data is moved to a register
1	Data is moved from a register

Table 10-2 CA Instruction Field Encodings (cont.)

C Field - CAU & DAU Condition Codes			
Condition	**Code**	**Flags**	**Meaning**
false	0 0 0 0 0 0	none	always false
true	0 0 0 0 0 1	none	always true
pl	0 0 0 0 1 0	n=0	result is nonnegative (plus)
mi	0 0 0 0 1 1	n=1	result is negative (minus)
ne	0 0 0 1 0 0	z=0	result not equal to zero
eq	0 0 0 1 0 1	z=1	result equal to zero
vc	0 0 0 1 1 0	v=0	overflow clear, no overflow
vs	0 0 0 1 1 1	v=1	overflow set, overflow
cc	0 0 1 0 0 0	c=0	carry clear, no carry
cs	0 0 1 0 0 1	c=1	carry set, carry
ge	0 0 1 0 1 0	n^v=0	greater than or equal to zero
lt	0 0 1 0 1 1	n^v=1	less than zero
gt	0 0 1 1 0 0	z|(n^v)=0	greater than zero
le	0 0 1 1 0 1	z|(n^v)=1	less than or equal to zero
hi	0 0 1 1 1 0	c|z=0	greater than (unsigned)
ls	0 0 1 1 1 1	c|z=1	less than (unsigned)
auc	0 1 0 0 0 0	U=0	underflow clear, no underflow
aus	0 1 0 0 0 1	U=1	underflow set, underflow
age	0 1 0 0 1 0	N=0	greater than or equal to zero
alt	0 1 0 0 1 1	N=1	less than zero
ane	0 1 0 1 0 0	Z=0	not equal to zero
aeq	0 1 0 1 0 1	Z=1	equal to zero
avc	0 1 0 1 1 0	V=0	overflow clear, no overflow
avs	0 1 0 1 1 1	V=1	overflow set, overflow
agt	0 1 1 0 0 0	N|Z=0	greater than zero
ale	0 1 1 0 0 1	N|Z=1	less than or equal to zero
	0 1 1 0 1 x		Reserved
	0 1 1 1 x x		Reserved

C Field - I/O Condition Codes			
Condition	**Code**	**Flags**	**Meaning**
ibe	1 0 0 0 0 0	ibf=0	input buffer empty
ibf	1 0 0 0 0 1	ibf=1	input buffer full
obf	1 0 0 0 1 0	obe=0	output buffer FULL
obe	1 0 0 0 1 1	obe=1	output buffer empty
	1 0 0 1 0 0		Reserved
	1 0 0 1 0 1		Reserved
	1 0 0 1 1 0		Reserved
	1 0 0 1 1 1		Reserved
syc	1 0 1 0 0 0	sy=0	SY (I/O sync) cleared
sys	1 0 1 0 0 1	sy=1	SY (I/O sync) set
fbc	1 0 1 0 1 0	fb=0	Serial I/O frame boundary clear
fbs	1 0 1 0 1 1	fb=1	Serial I/O frame boundary set
ir0c	1 0 1 1 0 0	ir0n=0	Interrupt Request 0 clear (logic 0)
ir0s	1 0 1 1 0 1	ir0n=1	Interrupt Request 0 set (logic 1)
ir1c	1 0 1 1 1 0	ir1n=0	Interrupt Request 1 clear (logic 0)
ir1s	1 0 1 1 1 1	ir1n=1	Interrupt Request 1 set (logic 1)
	1 1 x x x x		Reserved

Table 10-3 DA Instruction Field Encodings

DA - G Field	
G	Operation
0000	ic
0001	oc
0010	float16
0011	int16
0100	round
0101	ifalt
0110	ifaeq
0111	ifagt
1000	float32
1001	int32
1010	Reserved
1011	Reserved
1100	ieee (DSP to IEEE)
1101	dsp (IEEE to DSP)
1110	seed
1111	Reserved

DA - X, Y, Z Fields	
p	Operation
0000	Selects register-direct
0001	r1
0010	r2
0011	r3
0100	r4
0101	r5
0110	r6
0111	r7
1000	r8
1001	r9
1010	r10
1011	r11
1100	r12
1101	r13
1110	r14
1111	Not allowed

DA - X, Y, Z Fields	
i	Operation (p≠0000)
000	0
001	r15
010	r16
011	r17
100	r18
101	r19
110	-4(f), –2(i), –1(b)
111	+4(f), +2(i), +1(b)

DA - M Field	
M	Operation
000	a0
001	a1
010	a2
011	a3
100	0.0
101	1.0
110	Not Allowed
111	Not Allowed

DA - N Field	
N	Operation
00	a0
01	a1
10	a2
11	a3

DA - X,Y,Z Field	
M	Operation (p=0000)
000	a0 – X, Y fields only
001	a1 – X, Y fields only
010	a2 – X, Y fields only
011	a3 – X, Y fields only
100	Reserved
101	Reserved
110	Reserved
111	No write - Z field only

DA - S and F Fields	
S/F	Operation
0	+
1	–

10.2 Register Encodings

Tables 10-4 to 10-8 illustrate the bit-level encoding for the IO registers of the DSP3210. Tables 10-9 to 10-12 illustrate the bit-level encoding for the MMIO registers of the DSP3210.

Table 10-4 ps Register Encoding

Bit	15	14	13	12	11	10-	9	8	7	6	5	4	3	2	1	0
Field	0	0	ir1	ir0	fb	sy	obe	ibf	V	U	Z	N	c	v	z	n

Bit(s)	Field	Function
0	n	CAU **n** flag; means that the ALU result is negative.
1	z	CAU **z** flag; means that the ALU result is zero.
2	v	CAU **v** flag; means that the ALU result overflowed.
3	c	CAU **c** flag; means that the ALU result had a carry out of the MSB.
4	N	DAU **N** flag; means that the DAU result is negative.
5	Z	DAU **Z** flag; means that the DAU result is zero.
6	U	DAU **U** flag; means that the DAU result underflowed.
7	V	DAU **V** flag; means that the DAU result overflowed.
8	ibf	Serial input buffer full flag; means that the serial input buffer is full.
9	obe	Serial output buffer empty flag; means that the serial output buffer is empty.
10	sy	Serial sync set flag; means that the SY pin is high (logic one).
11	fb	Serial frame boundary flag; means that the SIO is at a frame boundary.
12	ir0	Pin IR0N is high (logic one).
13	ir1	Pin IR1N is high (logic one).
15—14	-	Reserved - read as zero.

Table 10-5 dauc Register Encoding

Bit	7	6	5 4	3 2 1 0
Field	0	COND	ROUND	CONV

Bit(s)	Field	Encoding	Function
3—0	CONV	x0x0	µ-law input conversion.
		x0x1	A-law input conversion.
		x1xx	Unsigned linear byte input conversion.
		0x0x	µ-law output conversion.
		0x1x	A-law output conversion.
		1xxx	Unsigned linear byte output conversion.
5—4	ROUND	x0	Round-to-nearest on float-to-integer conversions.
		01	Truncate towards -∞ on float-to-integer conversion.
		11	Truncate towards 0 on float-to-integer conversion.
6	COND	0	DA instructions *ifalt*, *ifaeq*, and *ifagt* have unconditional Z-writes.
		1	DA instructions *ifalt*, *ifaeq*, and *ifagt* have conditional Z-writes.
7	0	0	Must be set to zero; Read as zero.

Table 10-6 ctr Register Encoding

Bit	7	6	5	4	3	2	1	0
Field	0	0	N–5	N–4	N–3	N–2	N–1	N0

Bit(s)	Field	Meaning
0	N0	Current DAU **N** Flag
1	N-1	DAU **N** Flag computed for Instruction –1 (previous DA instruction that set flags)
2	N-2	DAU **N** Flag computed for Instruction –2
3	N-3	DAU **N** Flag computed for Instruction –3
4	N-4	DAU **N** Flag computed for Instruction –4
5	N-5	DAU **N** Flag computed for Instruction –5
7-6	-	Reserved - read as zero

Table 10-7 emr Register Encoding

Bit	15	14	132	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	EXT1	OFRM	IFRM	OBE	IBF	Res	TIMER	EXT0	Res	NAN	U/V	AERR	Reserved			SL

Each bit in this register corresponds to one of the maskable sources for an exception. A value of 1 in a position enables the corresponding interrupt source; a value of 0 disables the source. On reset all bits are set to 0 (zero).

Bit(s)	Field	Function	Priority
0	SL	If 1, select processor level to be visible on IACK[1—0]. Encoding: 00 - Base level 01 - Interrupt level 10 - Error level 11 - Double error level	-
1-3		Reserved	-
4	AERR	Addressing Error	Highest
5	U/V	DAU Underflow/Overflow	|
6	NAN	IEEE NAN	|
7		Reserved	|
8	EXT0	Interrupt pin 0 (IR0N) request	|
9	TIMER	TIMER counted to zero	|
10		Reserved	|
11	IBF	SIO input buffer full	|
12	OBE	SIO output buffer empty	|
13	IFRM	SIO input DMA frame complete	|
14	OFRM	SIO output DMA frame complete	V
15	EXT1	Interrupt pin 1 (IR1N) request	Lowest

Table 10-8 pcw Register Encoding

Bit	15	14	132	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	P/BN		BRC	D/SN	R/WN	C/PN	M/IN	B/LN	SBD	PBD			WB		WA	

Table 10-8 pcw Register Encoding (cont.)

Bit(s)	Field	Function
1,0	WA	These bits configure the number of wait-states that are generated for transactions in partition A. Set to {1,1} on reset.
		pcw 1,0: 00 = 0-or-more wait-states; 01 = 1-or-more wait-states; 10 = 2-or-more wait-states; **11 = 3-or-more wait-states**
3,2	WB	These bits configure the number of wait-states that are generated for transactions in partition B. Set to {1,1} on reset.
		pcw 3,2: 00 = 0-or-more wait-states; 01 = 1-or-more wait-states; 10 = 2-or-more wait-states; **11 = 3-or-more wait-states**
6—4	PBD	These bits configure the page size for asserting the page break detect (PBD) signal. A2—A9 are always masked, A17—A31 are always unmasked, A10—A16 masking is controlled by the PBD field when comparing the last address to external memory. Set to {0,0,0} on reset.

pcw 6,5,4	Page Size	DRAM size
000	**256 words**	**64k x n**
001	512 words	256k x n
010	1k words	1M x n
011	2k words	4M x n
100	4k words	16M x n
101	8k words	64M x n
110	16k words	256M x n
111	32k words	1G x n

7	SBD	This bit enables(1)/disables(0) sequential address break detection and can be used with a page size of 256 only. Set to one on reset.
8	B/LN	Configures the byte ordering. If 0, little endian byte ordering is selected. If 1, big endian byte ordering is selected. Set to one on reset.
9	M/IN	Configures the MS0—MS3 pins. If 0, Intel style signaling is selected. If 1, Motorola style signaling is selected. Set to one on reset.
10	C/PN	Selects between one of two memory maps. If 1, µComputer mode is selected and on-chip memory is at location zero. If 0, µProcessor mode is selected and external memory is at location zero. Reset value based on the state of bio[7] during reset.
11	R/WN	Configures the RW pin. If 0, RW pin is active high for write transactions. If 1, RW pin is active high for read transactions. Reset value based on the state of bio[6] during reset.
12	D/SN	Configures the operation of pins AEN/MRN and DEN/MWN. If 0, SRAM interface signaling selected (MRN and MWN). If 1, DRAM system bus interface signaling selected (AEN and DEN). Reset value based on the state of bio[5] during reset.
13	BRC	Reset value based on the state of bio[4] during reset. Controls Boot ROM routine. Also, when written with a one, modification of the pcw is disabled.
15—14	P/BN	Configures the PBD/BLMN signal. Set to {0,0} on reset.
		pcw 15,14: **00 = BLMN is selected**; 01 = PBD is selected; addresses to partition A are monitored. 10 = PBD is selected; addresses to partition B are monitored. 11 = PBD is selected; addresses to partitions A and B are monitored.

Table 10-9 ioc Register Encoding

Bit	31–24	23—20	19	18	17	16	15	14	13	12—10	9	8	7 6	5	4	3	2	1	0
Field	Res	ICN	OSZ	ISZ	OUT	IN	SAN	IIC	BO	OLEN	AOL	AOC	ILEN	AIL	AIC	SLEN		BC	ASY

Bit(s)	Field	Function
0	ASY	If 0, SY is external. If 1, SY is internal. When generated internally, SY = {ICK, OCK}/{256, 512, 1024}, based on ioc[1] and ioc[3,2].
1	BC	If 0, ICK is used to derive the internal load and SY signals. If 1, OCK is used to derive the internal load and SY signals.
3,2	SLEN	These bits select the frequency ratio of the on-chip load signal to the on-chip SY signal.
4	AIC	If 0, ICK is external. If 1, ICK is generated internally with a frequency specified by ioc[23—20].
5	AIL	If 0, ILD is external. If 1, ILD is generated internally with a frequency of {ICK,OCK}/32, based on ioc[1].
7,6	ILEN	These bits specify the length of the serial input data.
8	AOC	If 0, OCK is external. If 1, OCK is internally generated with a frequency specified by ioc[23—20].
9	AOL	If 0, OLD is external. If 1, OLD is internally generated: (1) with a frequency of {ICK,OCK}/32, based on ioc[1] if ioc[13]=0, or (2) in a burst-mode if ioc[13]=1.
12—10	OLEN	These bits specify the length of serial output data.
13	BO	If 0, the internal OLD is {ICK,OCK}/32, based on ioc[1]. If 1, the internal OLD is burst-mode, generated as a function of the status of the obuf and OSR.
14	IIC	If 0, the ICK is true (not inverted). If 1, the ICK is inverted.
15	SAN	If 0, clear sanity bit. If 1, set sanity bit.
16	IN	If 0, the LSB is received first during serial inputs. If 1, the MSB is received first during serial inputs.
17	OUT	If 0, the LSB is transmitted first during serial outputs. If 1, the MSB is transmitted first during serial outputs (can't be used with 24-bit output length).
18	ISZ	If 0, the input DMA transfer size is 32-bits. If 1, the input DMA size is specified by ILEN.
19	OSZ	If 0, the output DMA transfer size is 32-bits. If 1, the output DMA size is specified by OLEN.
23—20	ICN	These bits specify the divide-down rate of the internal clock generator. The internal clock generator has an additional fixed divide-down by four, i.e., if bits 23—20 represent N, the divide-down rate is N*4. If ICN = 0, the active clock generator is disabled.
31—24		Reserved; Read as zero.

SLEN:

ioc[3]	ioc[2]	
0	0	ratio = 32
0	1	ratio = 8
1	0	ratio = 16
1	1	ratio = 32

ILEN:

ioc[7]	ioc[6]	
0	0	input length is 32 bits (prior to ILD)
0	1	input length is 8 bits (after ILD)
1	0	input length is 16 bits (after ILD)
1	1	input length is 32 bits (after ILD)

OLEN:

ioc[12]	ioc[11]	ioc[10]	
0	0	0	reserved
0	0	1	output length is 8 bits
0	1	0	output length is 16 bits
0	1	1	output length is 32 bits
1	0	0	output length is 24 bits
1	0	1	reserved
1	1	x	reserved

Table 10-10 dmac Register Encoding

Bit	7	6	5	4	3	2	1	0
Field	IRB	IRS	IDE	ICS	ORB	ORS	ODE	OCS

Bit(s)	Field	Function
0	OCS	If 0, don't change output DMA settings, dmac[1,2], when the dmac register is loaded. If 1, change output DMA settings when the dmac register is loaded to new values specified in bits 1,2.
1	ODE	If 0, disable output DMA. If 1, enable output DMA and request DMA cycle when OBE is asserted, obuf is empty.
2	ORS	If 0, stop output DMA when the number of DMA transfers (OCOUNTER) reaches the number specified in OCNT. If 1, reload the current output DMA pointer and the current output transfer count registers, ODP and OCNT, respectively, when the number of DMA transfers (OCOUNTER) reaches the number specified in OCNT and keep output DMA enabled.
3	ORB	If 0, request output DMA cycles based on OBE and dmac[1]. If 1, request output DMA cycles based on OBE, dmac[1], **and** if the DSP3210 has ownership of the bus interface. If the DSP3210 is not the current bus master, the BRN is asserted. When the bus is granted, the DMA cycle is requested.
4	ICS	If 0, don't change input DMA settings, dmac[4,5], when the dmac register is loaded. If 1, change input DMA settings when the dmac register is loaded to new values specified in bits 4,5.
5	IDE	If 0, disable input DMA. If 1, enable input DMA and request DMA cycle when IBF is aserted, ibuf is full.
6	IRS	If 0, stop input DMA when the number of DMA transfers (ICOUNTER) reaches the number specified in ICNT. If 1, reload the current input DMA pointer and the current input transfer count registers, IDP and ICNT, respectively, when the number of DMA transfers (ICOUNTER) reaches the number specified in ICNT and keep output DMA enabled.
7	IRB	If 0, request input DMA cycles based on IBF and dmac[4]. If 1, request DMA cycles based on IBF, dmac[4], **and** if the DSP3210 has ownership of the bus interface. If the DSP3210 is not the current bus master, the BRN is asserted. When the bus is granted, the DMA cycle is requested.

Table 10-11 tcon Register Encoding

Bit	7	6	5	4	3	2	1	0
Field	SRC			OUT	H/LN	T/PN	R/SN	E/HN

Bit(s)	Field	Function
0	E/HN	Hold/enable count-down operation of counter 0 - Hold the count 1 - Enable the count-down operation of counter
1	R/SN	Select between continuous and one-time operation 0 - Stop counting when the count reaches zero 1 - Automatic reload of the counter from the initial count register when the count reaches zero
2	T/PN	Select pulse or toggle operation of the output when the count reaches zero 0 - For the external output, a pulse is generated when the count reaches zero 1 - For the external output, the output is the output of a toggle flip-flop which is toggled when the count reaches zero
3	H/LN	When tcon[2] is set to zero, pulse, select low active or high active pulse 0 - When tcon[2] is set to zero, the pulse is one CKI clock period wide and is low active (1->0->1) 1 - When tcon[2] is set to zero, the pulse is one CKI clock period wide and is active high (0->1->0)
4	OUT	If the BIO1 pin is configured as an output, 0 - the BIO1 pin has the value specified by the BIO1 field in the BIO register 1 - the BIO1 pin is selected to be the timer output
5—7	SRC	Selects the clock source for the timer tcon 7,6,5 000 Counter clock is CKI/2 001 Counter clock is CKI/4 010 Reserved 011 Reserved 100 Counter clock is from external source(pin BIO0) - rising edge 101 Counter clock is from external source (pin BIO0) - falling edge 11x Reserved

Table 10-12 bioc Register Encoding

Bit	7	6	5	4	3	2	1	0
Field	B7	B6	B5	B4	B3	B2	B1	B0

Each bit in the bioc register corresponds to one of the pins in the BIO port, B0—BIO0, B1—BIO1, ..., B7—BIO7. A value of 0 in the bit indicates that the associated pin is configured as an input; a value of 1 indicates that the pin is configured as an output. On reset all bits are set to zero (input).

Table 10-13 bio Register Encoding

Bit	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Field	BF7	BF6	BF5	BF4	BF3	BF2	BF1	BF0

Each pair of bits (field) in the bio register corresponds to one of the bio pins in the BIO port, BF0—BIO0, BF1—BIO1, ..., BF7—BIO7. When the bio register is written, the value of each output register bit is changed according to the following encoding of the bit field:

BF bit pair: bit 1,0

00	Keep the previous value in the bio output register bit
01	Set the bio output register bit to zero
10	Set the bio output register bit to one
11	Complement the value in the bio output register bit

The transfer function is described below, where Qt is the previous value and Qt+1 is the new value:

Qt	Bit1	Bit0	Qt+1	
0	0	0	0	previous
0	0	1	0	false
0	1	0	1	true
0	1	1	1	toggle
1	0	0	1	previous
1	0	1	0	false
1	1	0	1	true
1	1	1	0	toggle

When the bio register is read, the value of both the corresponding bio pin and the bio output register bit are read according to the following encoding of the bit field:

bit 0	Bio pin value
bit 1	Bio output register value

Chapter 11

Physical Specifications

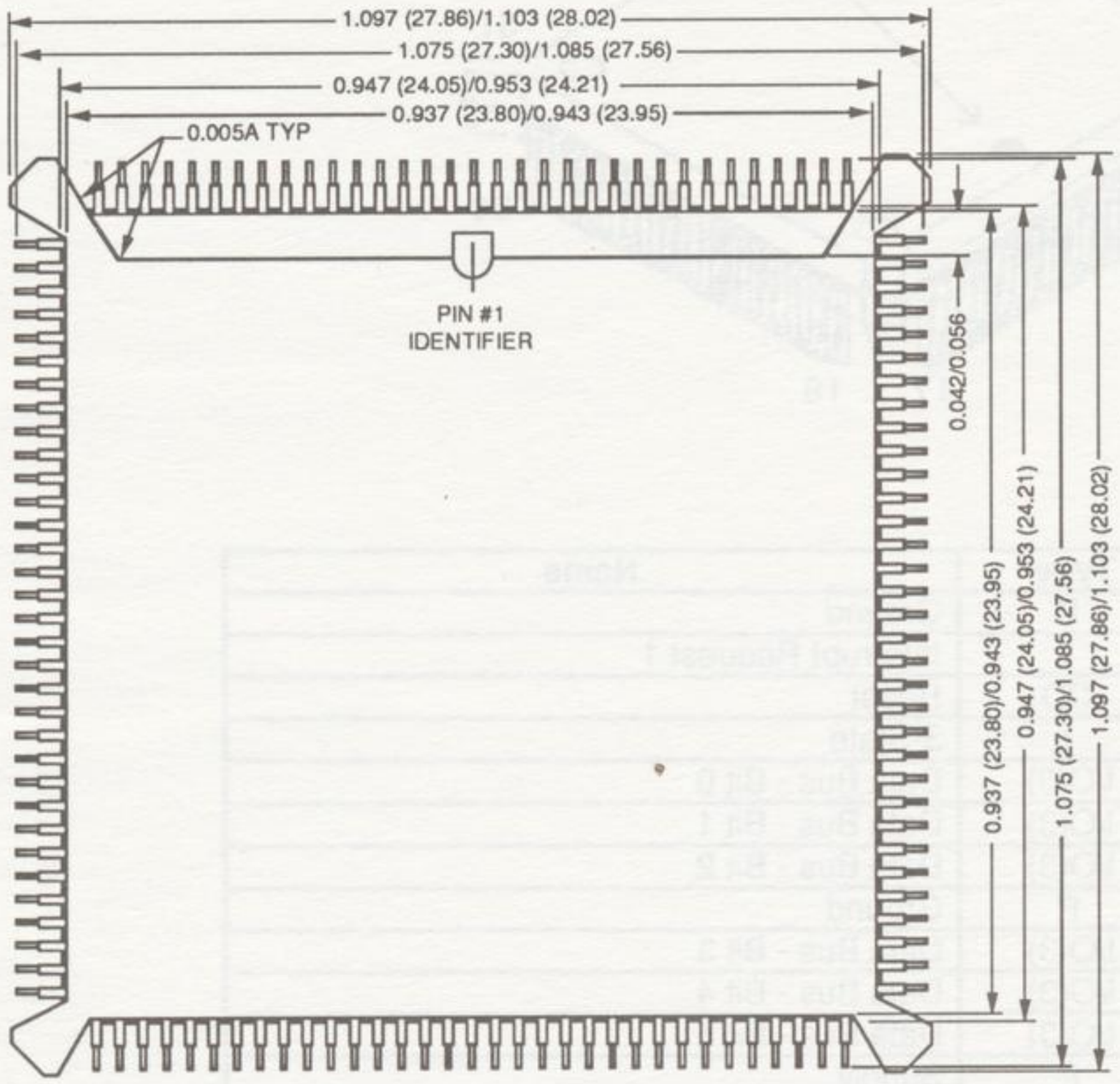
CHAPTER 11. PHYSICAL SPECIFICATIONS

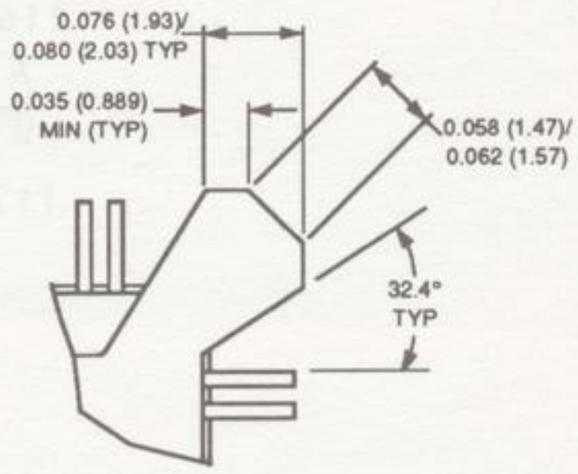
11. PHYSICAL SPECIFICATIONS 11-1
11.1 Pin Assignments 11-2
Table 11-1 Pin Descriptions 11-2

11. PHYSICAL SPECIFICATIONS

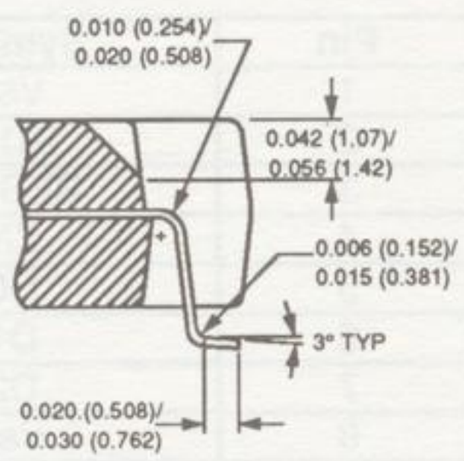
DSP3210 132-Pin PQFP (Plastic Quad Flat Package)

Dimensions are in inches and (millimeters).

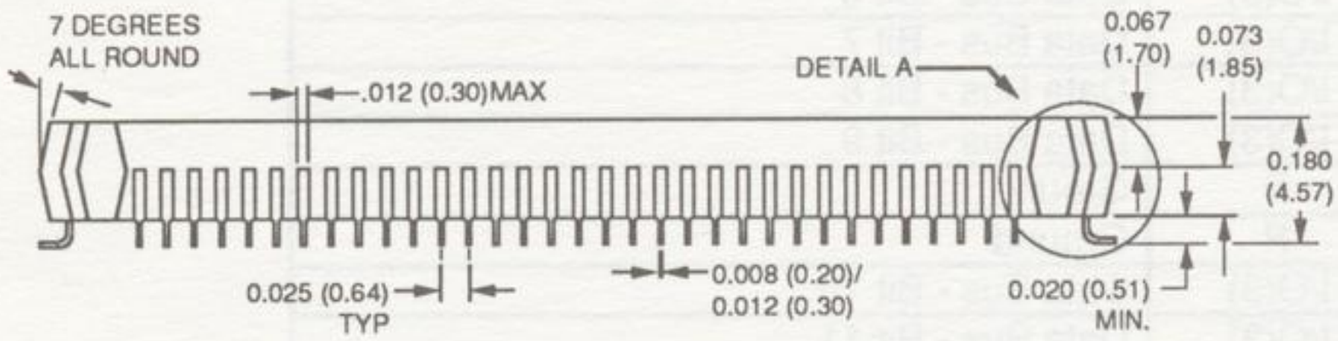




BUMPER DETAIL



DETAIL A



11.1 Pin Assignments

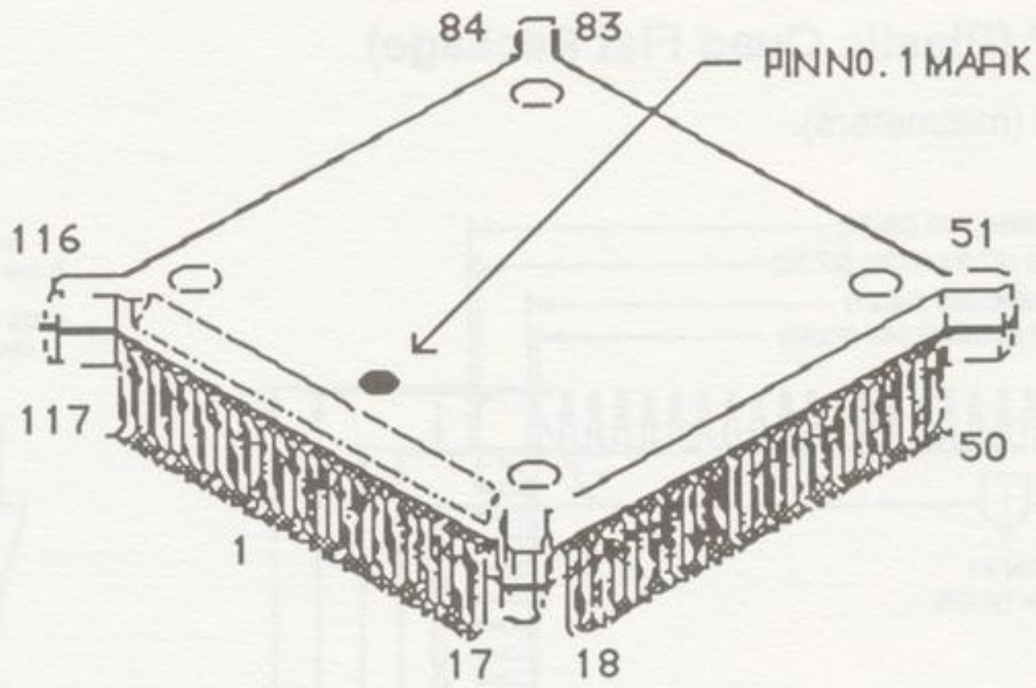


Table 11-1 Pin Descriptions

Pin	Symbol	Type*	Name
1	VSS	P	Ground
2	IR1N	I	Interrupt Request 1
3	RESTN	OD	Reset
4	ZN	I	3-State
5	D0	I/O(3)	Data Bus - Bit 0
6	D1	I/O(3)	Data Bus - Bit 1
7	D2	I/O(3)	Data Bus - Bit 2
8	VSS	P	Ground
9	D3	I/O(3)	Data Bus - Bit 3
10	D4	I/O(3)	Data Bus - Bit 4
11	D5	I/O(3)	Data Bus - Bit 5
12	VDD	P	Supply
13	D6	I/O(3)	Data Bus - Bit 6
14	D7	I/O(3)	Data Bus - Bit 7
15	D8	I/O(3)	Data Bus - Bit 8
16	D9	I/O(3)	Data Bus - Bit 9
17	VSS	P	Ground
18	VSS	P	Ground
19	D10	I/O(3)	Data Bus - Bit 10
20	D11	I/O(3)	Data Bus - Bit 11
21	D12	I/O(3)	Data Bus - Bit 12
22	D13	I/O(3)	Data Bus - Bit 13
23	VDD	P	Supply
24	D14	I/O(3)	Data Bus - Bit 14
25	D15	I/O(3)	Data Bus - Bit 15

* I = Input, O = Output, P = Power, OD = Open Drain, NC = No connection, (3) = 3-state.

Table 11-1 Pin Descriptions (continued)

Pin	Symbol	Type*	Name
26	DEN/MWN	I/O(3)	Data Enable/Write Strobe
27	Vss	P	Ground
28	DLE	I	Data Latch Enable
29	D16	I/O(3)	Data Bus - Bit 16
30	D17	I/O(3)	Data Bus - Bit 17
31	D18	I/O(3)	Data Bus - Bit 18
32	D19	I/O(3)	Data Bus - Bit 19
33	VDD	P	Supply
34	Vss	P	Ground
35	D20	I/O(3)	Data Bus - Bit 20
36	D21	I/O(3)	Data Bus - Bit 21
37	D22	I/O(3)	Data Bus - Bit 22
38	D23	I/O(3)	Data Bus - Bit 23
39	D24	I/O(3)	Data Bus - Bit 24
40	D25	I/O(3)	Data Bus - Bit 25
41	Vss	P	Ground
42	D26	I/O(3)	Data Bus - Bit 26
43	D27	I/O(3)	Data Bus - Bit 27
44	D28	I/O(3)	Data Bus - Bit 28
45	VDD	P	Supply
46	D29	I/O(3)	Data Bus - Bit 29
47	D30	I/O(3)	Data Bus - Bit 30
48	D31	I/O(3)	Data Bus - Bit 31
49	RW	O(3)	Read/Write
50	Vss	P	Ground
51	Vss	P	Ground
52	BGACKN	O(3)	Bus Grant Acknowledge
53	BRN	O(3)	Bus Request
54	ASN	O(3)	Address Strobe
55	CSN	O(3)	Cycle Start
56	BGN	I	Bus Grant
57	BERRN	I	Bus Error
58	SRDYN	I	Synchronous Ready
59	Vss	P	Ground
60	VDD	P	Supply
61	LOCKN	O(3)	Bus Lock
62	PBD/BLMN	O(3)	Page Break Detect/Block Move
63	MS3	O(3)	Memory Select 3
64	MS2	O(3)	Memory Select 2
65	MS1	O(3)	Memory Select 1
66	MS0	O(3)	Memory Select 0

* I = Input, O = Output, P = Power, OD = Open Drain, NC = No connection, (3) = 3-state.

Table 11-1 Pin Descriptions (continued)

Pin	Symbol	Type*	Name
67	Vss	P	Ground
68	A31	O(3)	Address Bus - Bit 31
69	A30	O(3)	Address Bus - Bit 30
70	A29	O(3)	Address Bus - Bit 29
71	A28	O(3)	Address Bus - Bit 28
72	A27	O(3)	Address Bus - Bit 27
73	VDD	P	Supply
74	Vss	P	Ground
75	A26	O(3)	Address Bus - Bit 26
76	A25	O(3)	Address Bus - Bit 25
77	A24	O(3)	Address Bus - Bit 24
78	A23	O(3)	Address Bus - Bit 23
79	A22	O(3)	Address Bus - Bit 22
80	A21	O(3)	Address Bus - Bit 21
81	A20	O(3)	Address Bus - Bit 20
82	A19	O(3)	Address Bus - Bit 19
83	Vss	P	Ground
84	Vss	P	Ground
85	AEN/MRN	I/O(3)	Address Enable/Read Strobe
86	A18	O(3)	Address Bus - Bit 18
87	VDD	P	Supply
88	A17	O(3)	Address Bus - Bit 17
89	A16	O(3)	Address Bus - Bit 16
90	A15	O(3)	Address Bus - Bit 15
91	A14	O(3)	Address Bus - Bit 14
92	Vss	P	Ground
93	A13	O(3)	Address Bus - Bit 13
94	A12	O(3)	Address Bus - Bit 12
95	A11	O(3)	Address Bus - Bit 11
96	A10	O(3)	Address Bus - Bit 10

* I = Input, O = Output, P = Power, OD = Open Drain, NC = No connection, (3) = 3-state.

Table 11-1 Pin Descriptions (continued)

Pin	Symbol	Type*	Name
97	A9	O(3)	Address Bus - Bit 9
98	VDD	P	Supply
99	Vss	P	Ground
100	A8	O(3)	Address Bus - Bit 8
101	A7	O(3)	Address Bus - Bit 7
102	A6	O(3)	Address Bus - Bit 6
103	A5	O(3)	Address Bus - Bit 5
104	A4	O(3)	Address Bus - Bit 4
105	A3	O(3)	Address Bus - Bit 3
106	Vss	P	Ground
107	A2	O(3)	Address Bus - Bit 2
108	CKI	I	Clock In
109	BIO7	I/O(3)	Bit I/O - Bit 7
110	BIO6	I/O(3)	Bit I/O - Bit 6
111	VDD	P	Supply
112	BIO5	I/O(3)	Bit I/O - Bit 5
113	BIO4	I/O(3)	Bit I/O - Bit 4
114	BIO3	I/O(3)	Bit I/O - Bit 3
115	BIO2	I/O(3)	Bit I/O - Bit 2
116	Vss	P	Ground
117	Vss	P	Ground
118	BIO1	I/O(3)	Bit I/O - Bit 1
119	BIO0	I/O(3)	Bit I/O - Bit 0
120	SY	I/O(3)	Sync
121	ICK	I/O(3)	Input Clock
122	VDD	P	Supply
123	DI	I	Serial Data In
124	DO	O(3)	Serial Data Out
125	OCK	I/O(3)	Output Clock
126	Vss	P	Ground
127	ILD	I/O(3)	Input Load
128	OLD	I/O(3)	Output Load
129	IACK0	O(3)	Interrupt Acknowledge 0
130	IACK1	O(3)	Interrupt Acknowledge 1
131	IR0N	I	Interrupt Request 0
132	VDD	P	Supply

* I = Input, O = Output, P = Power, OD = Open Drain, NC = No connection, (3) = 3-state.

Chapter 12

Electrical Specifications

CHAPTER 12. ELECTRICAL SPECIFICATIONS

12. ELECTRICAL AND TIMING SPECIFICATIONS 12-1
12.1 Electrical Specifications 12-1
Table 12-1 Electrical Requirements and Characteristics 12-1
12.1.1 Maximum Ratings 12-1
12.2 Timing Specifications 12-2
Figure 12-1 CKI Timing Specifications 12-3
Table 12-2 Timing Requirements for CKI 12-3
Table 12-3 Timing Requirements for Synchronous Bus Interface Inputs 12-3
Table 12-4 Timing Characteristics for Synchronous Bus Interface Outputs 12-4
Table 12-5 Timing Relationships for Synchronous Bus Interface Operation 12-5
Figure 12-2 Synchronous Bus Interface Timing 12-6
Figure 12-3 Asynchronous Bus Interface Timing 12-7
Table 12-6 Timing Relationships for Asynchronous Bus Interface Operation 12-7
Figure 12-4 Bus Arbitration Timing 12-8
Table 12-7 Timing Characteristics for Bus Arbitration 12-8
Figure 12-5 Serial Input Timing 12-9
Table 12-8 Timing Requirements for Serial Inputs 12-9
Figure 12-6 Serial Output Timing 12-10
Table 12-9 Timing Requirements for Serial Outputs 12-10
Table 12-10 Timing Characteristics for Serial Outputs 12-10
Table 12-11 Timing Requirements for Serial Clock Generation 12-11
Table 12-12 Timing Characteristics for Serial Clock Generation 12-11
Figure 12-7 ac Serial Clock Generator Timing 12-12
Figure 12-8 Bit I/O Timing 12-13
Table 12-13 Timing Requirements for Bit I/O 12-13
Table 12-14 Timing Characteristics for Bit I/O 12-13
Figure 12-9 Interrupt Timing 12-14
Table 12-15 Timing Requirements for Interrupts 12-14
Table 12-16 Timing Characteristics for Interrupts 12-14
Figure 12-10 RESTN 12-15
Table 12-17 Timing Requirements for Reset 12-15
Table 12-18 Timing Characteristics for Reset and ZN 12-15
Figure 12-11 Power-on Reset Timing 12-16
Figure 12-12 Externally Controlled Reset and ZN Timing 12-16

12. ELECTRICAL AND TIMING SPECIFICATIONS

12.1 Electrical Specifications

Table 12-1 Electrical Requirements and Characteristics

The parameters below are valid for the following conditions: T_A = 0 to 70 °C; V_{DD} = 5 V ± 5%; V_{SS} = 0 V

Description		Symbol	Min	Max	Unit
Input Voltage except CKI:	Low	VIL	—	0.8	V
	High	VIH	2.0	—	V
Input Voltage for CKI:	Low	VILC	—	0.4	V
	High	VIHC	2.4	—	V
Output Voltage:	(IOL = 6 mA) Low (TTL)	VOL	—	0.4	V
	(IOL = 5 µA) Low (CMOS)	VOL	—	0.2	V
	(IOL = –2 mA) High (TTL)	VOH	2.4	—	V
	(IOL = –5 µA) High (CMOS)	VOH	V_{DD} – 0.2	—	V
Output Current:	(VOL = 0.4 V) Low	IOL	—	6.0	mA
	(VOH = 2.4 V) High	IOH	–2.0	—	mA
Input Leakage except ZN:	a	IIL	–5	—	µA
	a	IIH	—	5	µA
Input Leakage for ZN:	(VIL = 0 V) Low	IIL	–100	—	µA
	(VIH = 5.5V) High	IIH	—	5	µA
Output Hi-Z Current:	(Vapplied = 0.0 V) Low	IOZL	–10	—	µA
	(Vapplied = 5.25 V) High	IOZH	—	10	µA
Input, Output, IOput capacitance		CI	—	10	pF
Power Supply Current*	V_{DD} = 5.25 V, T = 18 ns	IDD	—	220	mA
Power Dissipation†	V_{DD} = 5.25 V, T = 18 ns	PD	—	1.20	W

* Current in the input buffers is highly dependent on the input voltage level. At full CMOS levels, essentially no DC current is drawn, but for levels near the threshhold of 1.4 V, high and unstable levels of current may flow. There are 56 input buffers on the chip. If all inputs are connected to a dc voltage of 1.4 V, an additional current in the range of 100 mA can be drawn.

† The power dissipation listed is for internal power dissipation only. Total power dissipation can be calculated based on the system configuration by adding C * V_{DD}^2 * f for each output, where C is the load capacitance, and f is the output frequency.

12.1.1 Maximum Ratings

Stresses in excess of the Absolute Maximum Ratings can cause permanent damage to the device. These are absolute stress ratings only. Functional operation of the device is not implied at these or any other conditions in excess of those given in the operational sections of the data sheet. Exposure to Absolute Maximum Ratings for extended periods can adversely affect device reliability.

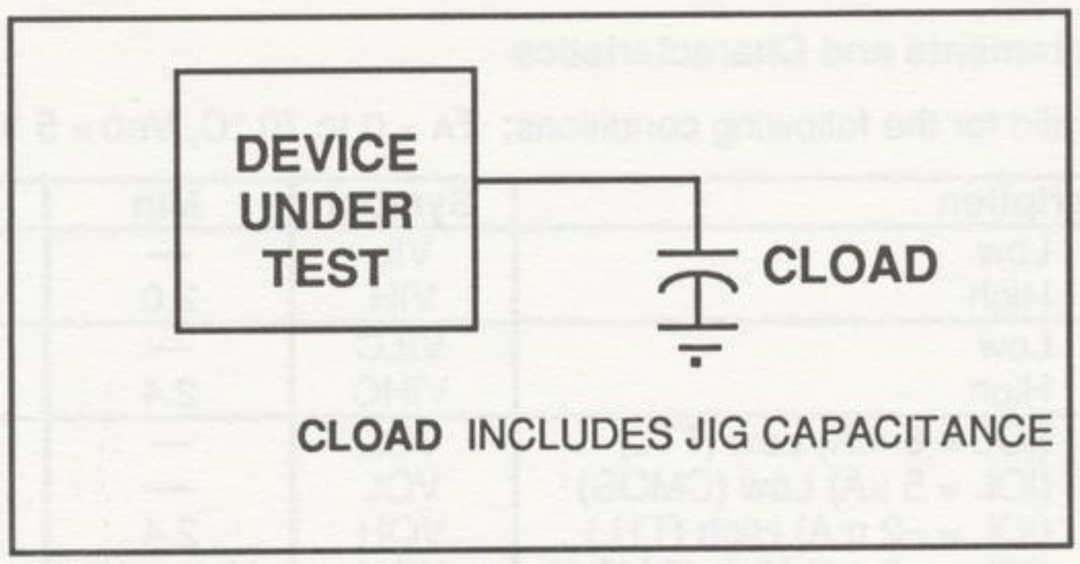
VOLTAGE RANGE ON ANY PIN WITH RESPECT TO GROUND .....–0.5 to + 6 V
AMBIENT TEMPERATURE RANGE.....–40 to +120 °C
STORAGE TEMPERATURE RANGE ..... –65 to +150 °C

Warning: All CMOS devices are prone to latch-up if excessive current is injected to/from the substrate. To prevent latch-up at power-up, no input pin should be subjected to input voltages greater than VIL, or less than V_{SS} – 0.5 V before V_{DD} is applied. After power-up, input should not be greater than V_{DD} + 0.5 V or less than V_{SS} – 0.5 V.

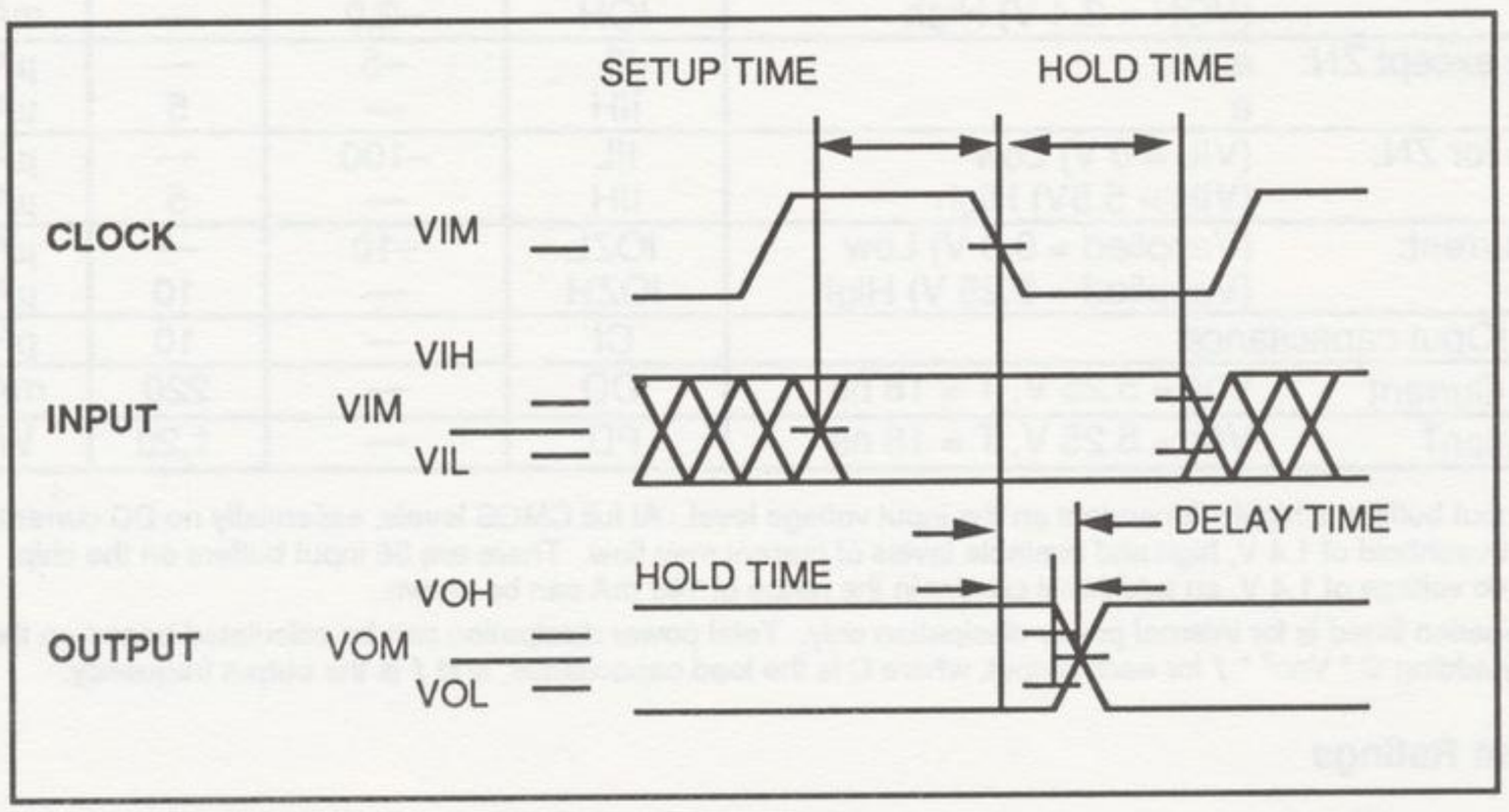
Handling Precautions: All MOS devices must be handled with certain precautions to avoid damage due to the accumulation of static charge. **Although input protection circuitry has been incorporated into the devices to minimize the effect of this static buildup, proper precautions should be taken to avoid exposure to electrostatic discharge during handling and mounting.** AT&T employs a human body model for ESD susceptibility testing. Since the failure voltage of electronic devices is dependent on the current and voltage and, hence, the resistance and capacitance, it is important that standard values be employed to establish a reference by which to compare test data. Values of 100 pF and 1500 Ω are the most common and are the values used in the AT&T human body model test circuit. The breakdown voltage for the DSP3210 is greater than 2000 V, according to the human body model. ESD data for the charge device model is available on request.

12.2 Timing Specifications

The characteristics listed are valid under the following conditions: V_{DD} = 5 V ± 5%; V_{SS} = 0 V; T_A = 0 to 70 °C.



ac Testing Load Circuit



ac Testing Input, Output Waveform

Test conditions for inputs (unless otherwise specified):

- Rise and fall times of 4 ns or less
- Timing reference level for setup times is VIM = 1.5V
- Timing reference levels for hold times is VIH, VIL

Test conditions for outputs (unless otherwise specified):

- CLOAD = 50 pF unless otherwise stated
- Timing reference level for delay times is VOM = 1.5 V
- Timing reference levels for hold times is VOH, VOL

Tables 12-2 to 12-18 are the timing specifications for the 55 MHz DSP3210.

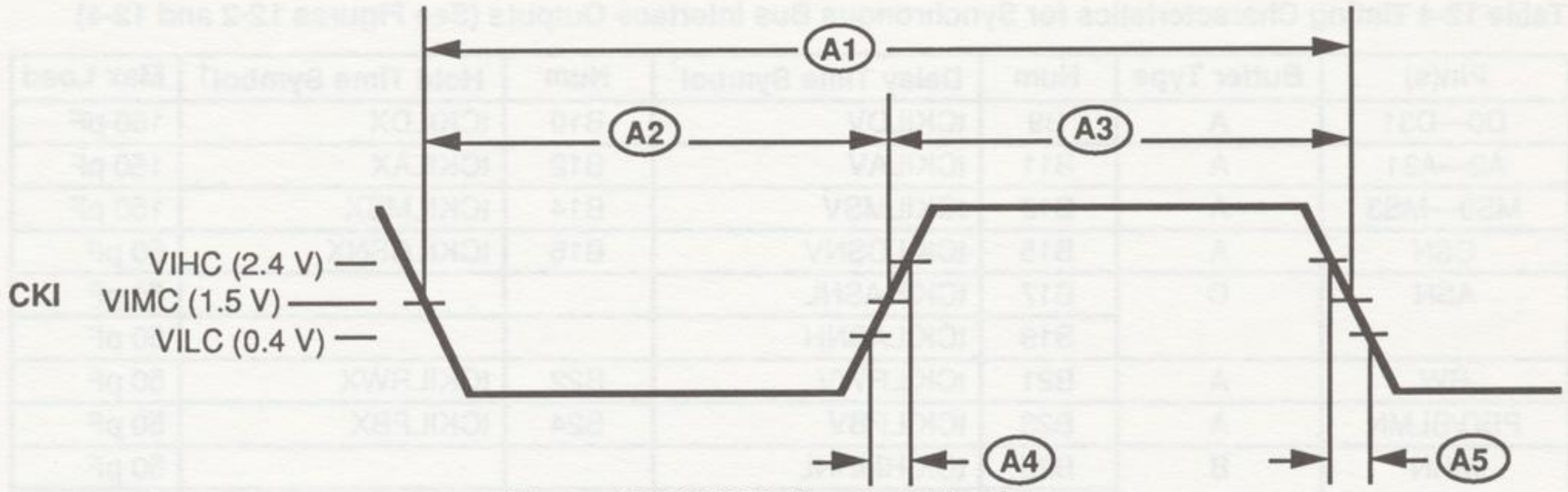


Figure 12-1 CKI Timing Specifications

Table 12-2 Timing Requirements for CKI (See Figure 12-1)

Num*	Symbol	Description	55 MHz		
			Min	Max	Unit
A1	tCKILCKIL	Clock In Period†	18	—‡	ns
A2	tCKILCKIH	Clock In Low	8.1	—	ns
A3	tCKIHCKIL	Clock In High	8.1	—	ns
A4	tCKIRISE	Clock In Rise Time	—	4	ns
A5	tCKIFALL	Clock In Fall Time	—	4	ns

* This column in all timing requirements and/or characteristic tables refers to the internal alpha-numeric symbol in the timing diagrams.
† tCKILCKIL is referred to as T for specifications that are a function of the period of CKI.
‡ This device is static. However, the maximum period guaranteed by test is 50 ns.

Table 12-3 Timing Requirements for Synchronous Bus Interface Inputs (See Figures 12-2 and 12-4)

Num	Symbol	Description	Min	Max	Unit
B1	tSRNVCKIL	SRDYN Setup to CKI Low	6	—	ns
B2	tCKILSRNX	SRDYN Hold after CKI Low	3	—	ns
B3	tBENVCKIL	BERRN Setup to CKI Low	6	—	ns
B4	tCKILBRNX	BERRN Hold after CKI Low	3	—	ns
B5	tBGNVCKIL	BGN Setup to CKI Low	6	—	ns
B6	tCKILBGNX	BGN Hold after CKI Low	3	—	ns
B7	tDVCKIL	DATAin Setup to CKI Low	0	—	ns
B8	tCKILDX	DATAin Hold after CKI Low	6	—	ns

Table 12-4 Timing Characteristics for Synchronous Bus Interface Outputs (See Figures 12-2 and 12-4)

Pin(s)	Buffer Type	Num	Delay Time Symbol*	Num	Hold Time Symbol†	Max Load
D0—D31	A	B9	tCKILDV	B10	tCKILDX	150 pF
A2—A31	A	B11	tCKILAV	B12	tCKILAX	150 pF
MS0—MS3	A	B13	tCKILMSV	B14	tCKILMSX	150 pF
CSN	A	B15	tCKILCSNV	B16	tCKILCSNX	50 pF
ASN	C	B17	tCKIHASNL			50 pF
		B19	tCKILASNH			50 pF
RW	A	B21	tCKILRWV	B22	tCKILRWX	50 pF
PBD/BLMN	A	B23	tCKILPBV	B24	tCKILPBX	50 pF
MRN	B	B25	tCKIHMRNL			50 pF
		B27	tCKILMRNH			50 pF
MWN	B	B29	tCKIHMWNV			50 pF
LOCKN	A	B31	tCKILLONV	B32	tCKILLONX	50 pF
BRN	A	B33	tCKILBRNV	B34	tCKILBRNX	50 pF
BGACKN	B	B35	tCKILBGANL	B36	tCKILBGANHX	50 pF
	A	B37	tCKIHBGANH	B38	tCKIHBGANLX	50 pF

* Description for all Delay Time Symbols in this table is: CKI (Low,High) to Signal (Valid, High, Low).
† Description for all Hold Time Symbols in this table is: Signal Value Held after CKI (Low, High).

Synchronous Delay Times (B9,B11,...B35,B37)		Min			Max			Unit
Buffer Type		A	B	C	A	B	C	
VOM	50 pF	—	—	—	15	14	13	ns
	100 pF	—	NA	NA	18	NA	NA	ns
	150 pF	—	NA	NA	21	NA	NA	ns
VOL	50 pF	—	—	—	17	16	15	ns
	100 pF	—	NA	NA	22	NA	NA	ns
	150 pF	—	NA	NA	27	NA	NA	ns
VOH	50 pF	—	—	—	17	16	15	ns
	100 pF	—	NA	NA	20	NA	NA	ns
	150 pF	—	NA	NA	23	NA	NA	ns
Synchronous Hold Times (B10,B12,...B36,B38)		**Min**			**Max**			
Buffer Type		A	B	C	A	B	C	Unit
VOL	50,100,150 pF	5	4	4	—	—	—	ns
VOH	50,100,150 pF	5	5	5	—	—	—	ns

Note: NA = Not Applicable.

Table 12-5 Timing Relationships for Synchronous Bus Interface Operation (See Figure 12-2)

Note: **All** signals referred to in this table are equally loaded with 50 pF. Signal timing relationships with respect to DATAin supercede B7 and B8 timing requirements.

Num	Symbol	Description	Min	Max	Unit
B39	tCKILDE	Data Bus Low-Z after CKI Low	0	—	ns
B40	tCKILDZ	Data Bus Hi-Z after CKI Low	—	12	ns
B41	tAVDINV	Address* Valid to DATAin Valid	—	2 * T – 11 + N * T	ns
B42	tAVASNL	Address Valid to Address Strobe Low	0.5 * T – 3†	0.5 * T	ns
B43	tASNLASNH	Address Strobe Width	1.5 * T – 2 + N * T	—	ns
B44	tASNHAX	Address Hold after Address Strobe High	0	—	ns
B45	tASNLDINV	Address Strobe Low to DATAin Valid	—	1.5 * T – 9 + N * T	ns
B46	tDINASNH	DATAin Setup to Address Strobe High	8	—	ns
B47	tASNHDX	DATAin Hold after Address Strobe High	–2	—	ns
B48	tASNHDZ	Data Bus Hi-Z after Address Strobe High	–3.0	2	ns
B49	tAVMRNL	Address Valid to Read Strobe Low	0.5 * T – 4	—	ns
B50	tMRNLMRNH	Read Strobe Width	1.5 * T – 4 + N * T	—	ns
B51	tMRNHMRNL	Read Strobe High to Read Strobe Low	0.5*T – 2	—	ns
B52	tMRNLDINV	Read Strobe Low to DATAin Valid	—	1.5 * T – 11 + N * T	ns
B53	tDINVMGNH	DATAin Setup to Read Strobe High	10	—	ns
B54	tMRNHDX	DATAin Hold after Read Strobe High	–4	—	ns
B55	tMRNHDE	Read Strobe High to Data Bus Low-Z	T – 4	—	ns
B56	tRWVDE	Read-Write Asserted to Data Bus Low-Z	T – 4	—	ns
B57	tAVMWNL	Address Valid to Write* Strobe Low	0.5 * T – 4	—	ns
B58	tMWNLMWNH	Write Strobe Width	T + M * T‡	—	ns
B59	tMWNHAX	Address Hold after Write Strobe High	0.5 * T – 4	—	ns
B60	tDVMWNH	Data Bus Valid to Write Strobe High: 0—1 Configured Wait-States 2—3 Configured Wait-States	 0.5 * T – 2 0.5 * T – 2 + (M – 1) * T	 — —	 ns ns
B61	tMWNHDZ	Data Bus Hi-Z after Write Strobe High	0.5 * T – 4	0.5 * T + (N – M) * T + 2	ns
B77	tASNLDV	Address Strobe Low to Data Bus Valid: 0 ws 1—3 ws	 — —	 0.5 * T + 2 1.5 * T + 2	 ns ns

* Address refers to A2—A31 and MS0—MS3.
† T = tCKILCKIL; N = number of wait-states (programmed plus external).
‡ M = Number of wait-states configured in pcw.

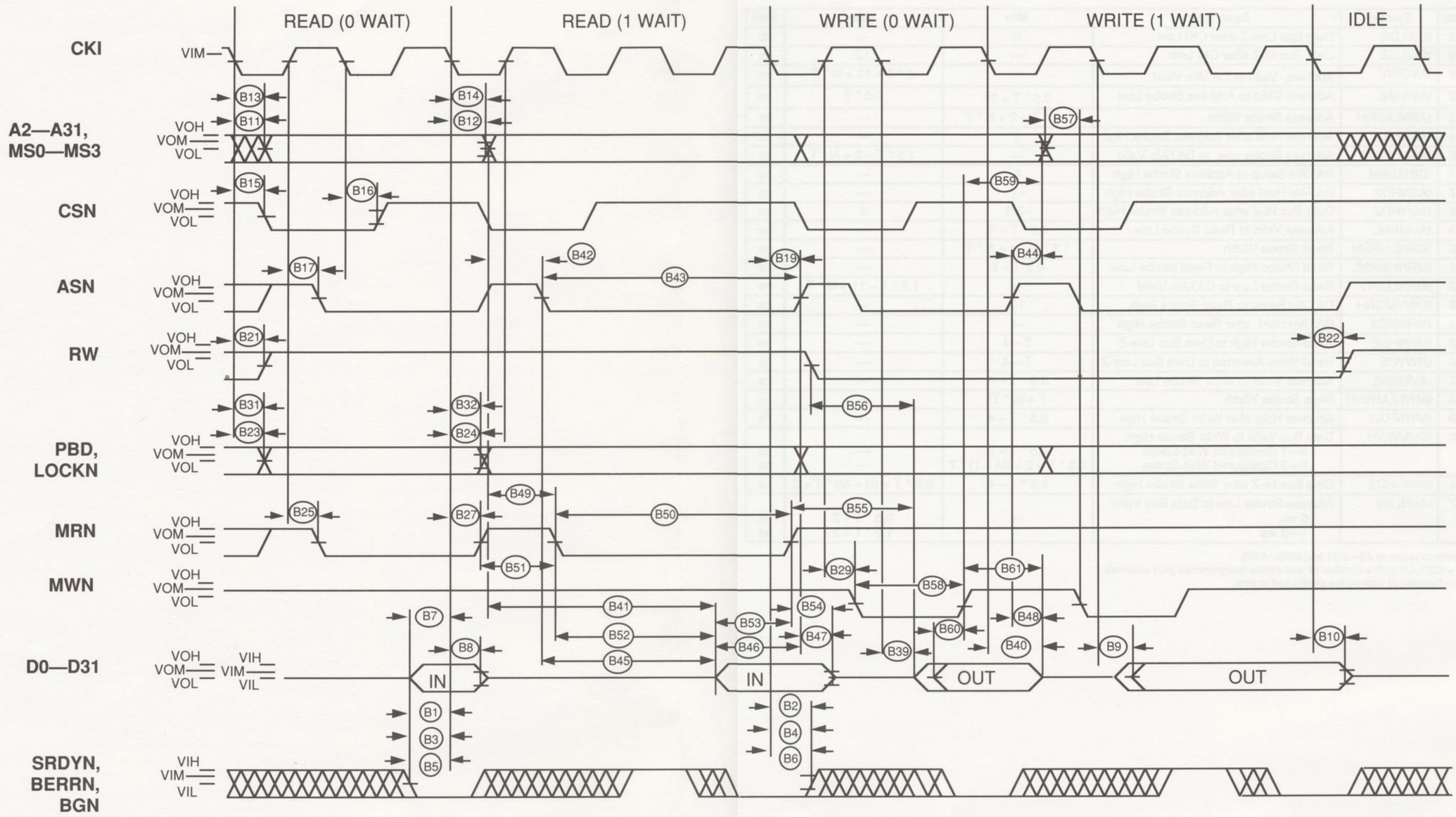


Figure 12-2 Synchronous Bus Interface Timing

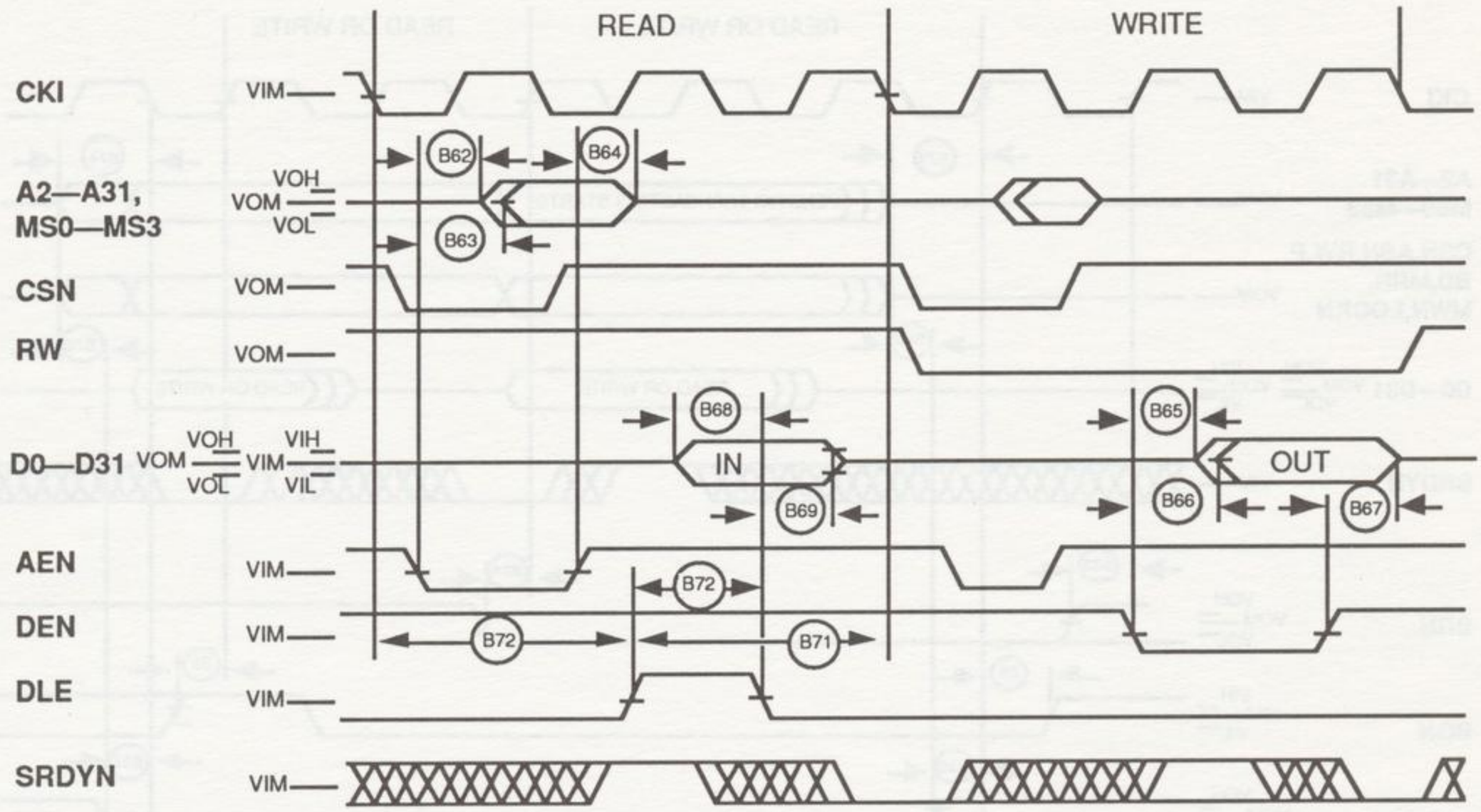


Figure 12-3 Asynchronous Bus Interface Timing

Table 12-6 Timing Relationships for Asynchronous Bus Interface Operation (See Figure 12-3)

Num	Symbol	Description	Min	Max	Unit
B62	tAENLAE	Address* Enable Low to Address Low-Z	0	—	ns
B63	tAENLAV: 50 pF 100 pF 150 pF	Address Enable Low to Address Valid	— — —	12 16 19	ns ns ns
B64	tAENHAZ	Address Enable High to Address High-Z	—	7	ns
B65	tDENLDE	Data Enable Low to Data Bus Low-Z	0	—	ns
B66	tDENLDV: 50 pF 100 pF 150 pF	Data Enable Low to Data Bus Valid	— — —	12 16 19	ns ns ns
B67	tDENHDZ	Data Enable High to Data Bus High-Z	—	7	ns
B68	tDVDLEL	DATAin Setup to Data Latch Enable Low	2	—	ns
B69	tDVDLEL	DATAin Hold after Data Latch Enable Low	4	—	ns
B70	tDLEHDLEL	Data Latch Enable Width	8	—	ns
B71	tDLEHCKIL	Data Latch Enable High to CKI Low when Asserted during State Ending Read Transaction (Bus Interface State Machine is in State E and SRDYN is Asserted)	3	—	ns
B72	tCKILDLEH	CKI Low to Data Latch Enable High after State Ending Read Transaction	6	—	ns

* Address refers to A2—A31 and MS0—MS3.

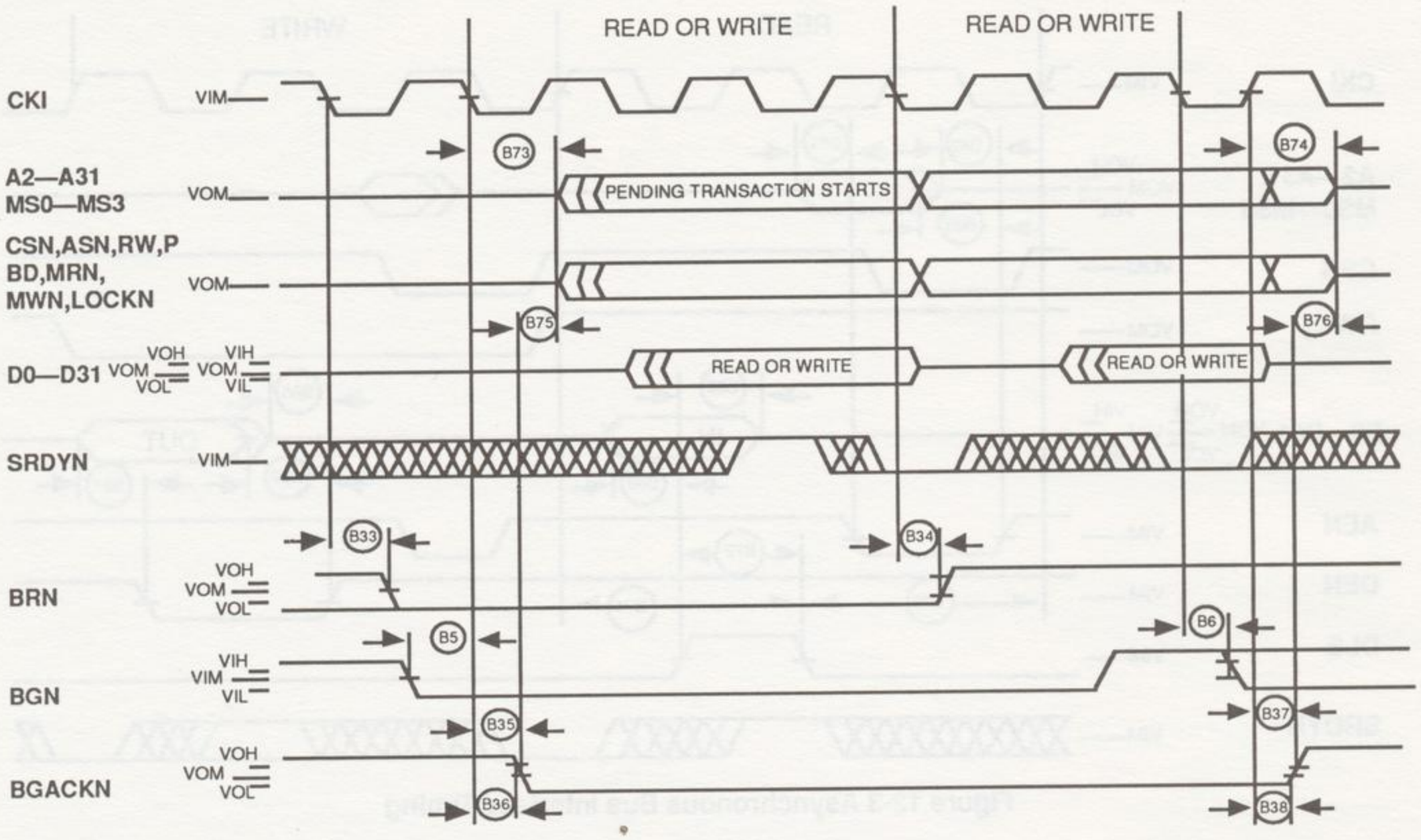


Figure 12-4 Bus Arbitration Timing

Table 12-7 Timing Characteristics for Bus Arbitration (See Figure 12-4)

Num	Symbol	Description	Min	Max	Unit
B73	tCKILBISE	CKI Low to Bus Interface Signals* Low-Z	4	—	ns
B74	tCKILDZ	CKI High to Bus Interface Signals High-Z	4	12	ns
B75	tBGANLBISE	BGACKN Low to Bus Interface Signals Low-Z	–4	—	ns
B76	tBGANHBISZ	BGACKN High to Bus Interface Signals High-Z	–4	2	ns

* Bus Interface Signals are defined to be A2—A31, MS0—MS3,CSN,ASN,RW,PBD,MRN,MWN, and LOCKN.

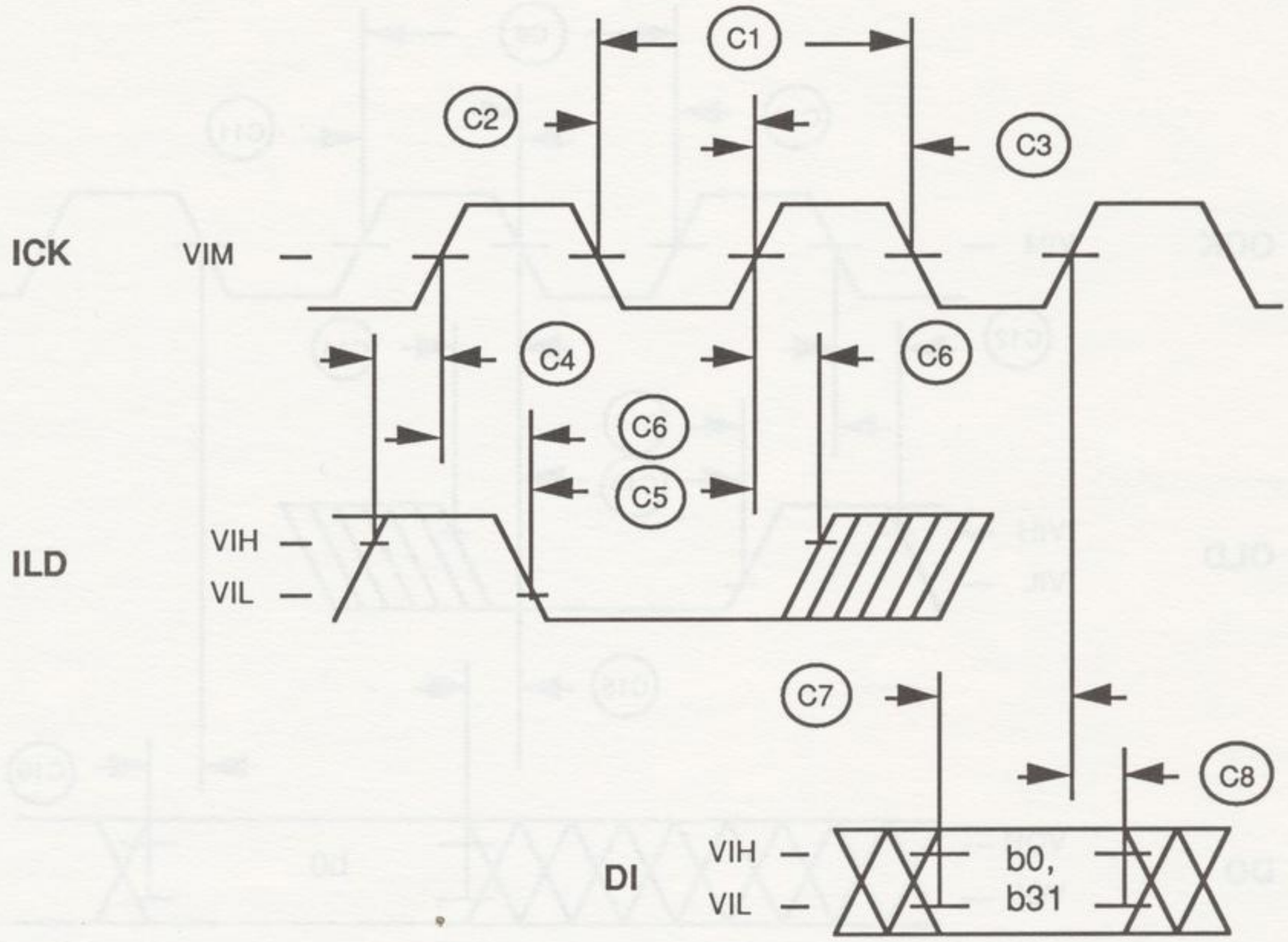


Figure 12-5 Serial Input Timing

Table 12-8 Timing Requirements for Serial Inputs (See Figure 12-5)

Num	Symbol	Description	Min	Max	Unit
C1	tICKLICKL	Serial Input Clock Period	40	—*	ns
C2	tICKLICKH	Serial Input Clock Low Time	18	—	ns
C3	tICKHICKL	Serial Input Clock High Time	18	—	ns
C4	tILDHICKH	Input Load High Setup Time	8	—	ns
C5	tILDLICKH	Input Load Low Setup Time	8	—	ns
C6	tICKHILDX	Input Load Hold Time	0	—	ns
C7	tDIVICKH	Serial Input Data Setup	8	—	ns
C8	tICKHDIX	Serial Input Data Hold	0	—	ns

* This device is static. However, the maximum period guaranteed by test is 200 ns.

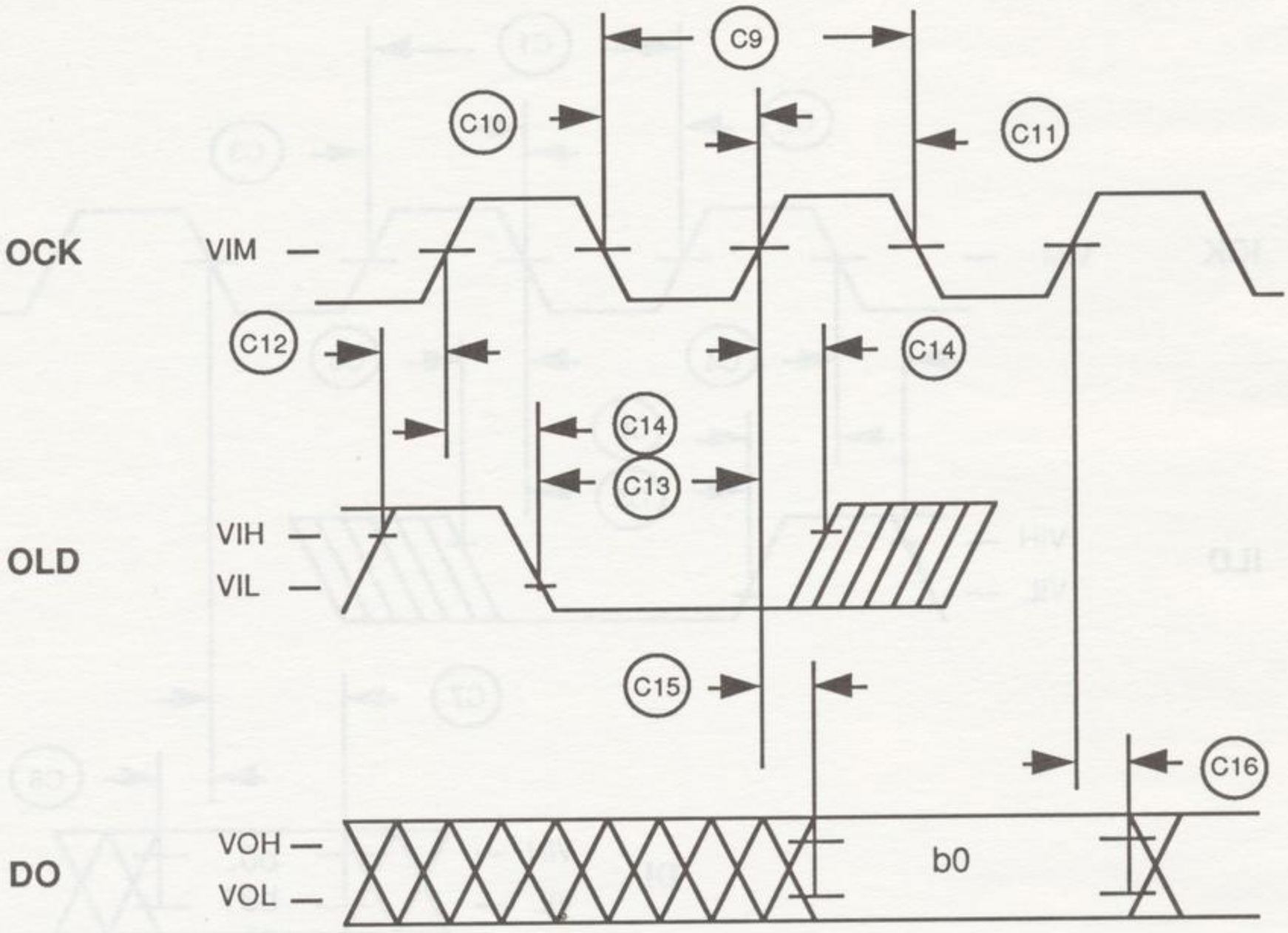


Figure 12-6 Serial Output Timing

Table 12-9 Timing Requirements for Serial Outputs (See Figure 12-6)

Num	Symbol	Description	Min	Max	Unit
C9	tOCKLOCKL	Serial Output Clock Period	40	—*	ns
C10	tOCKLOCKH	Serial Output Clock Low Time	18	—	ns
C11	tOCKHOCKL	Serial Output Clock High Time	18	—	ns
C12	tOLDHOCKH	Output Load High Setup Time	8	—	ns
C13	tOLDLOCKH	Output Load Low Setup Time	8	—	ns
C14	tOCKHOLDX	Output Load Hold Time	0	—	ns

* This device is static. However, the maximum period guaranteed by test is 200 ns.

Table 12-10 Timing Characteristics for Serial Outputs (See Figure 12-6)

Num	Symbol	Description	Min	Max	Unit
C15	tOCKHDOV	Serial Output Data Delay	—	23	ns
C16	tOCKHDOX	Serial Output Data Hold	2	—	ns

Table 12-11 Timing Requirements for Serial Clock Generation (See Figure 12-7)

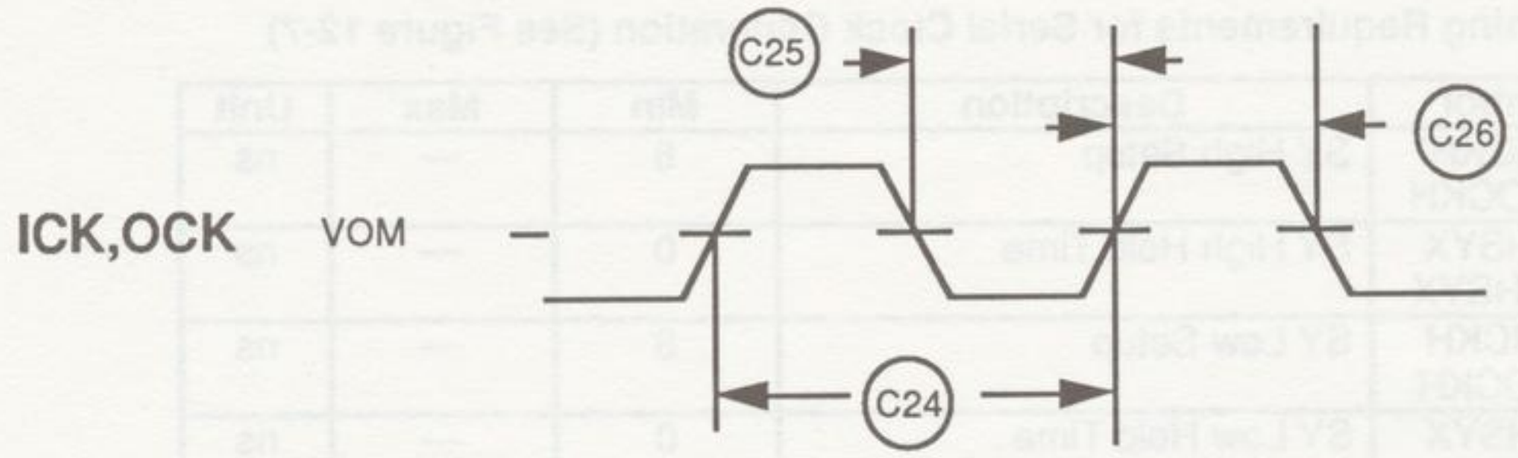
Num	Symbol	Description	Min	Max	Unit
C17	tSYHICKH* tSYHOCKH	SY High Setup	8	—	ns
C18	tICKHSYX tOCKHSYX	SY High Hold Time	0	—	ns
C19	tSYLICKH tSYLOCKH	SY Low Setup	8	—	ns
C20	tICKHSYX tOCKHSYX	SY Low Hold Time	0	—	ns

* ICK or OCK as selected by IOC[1].

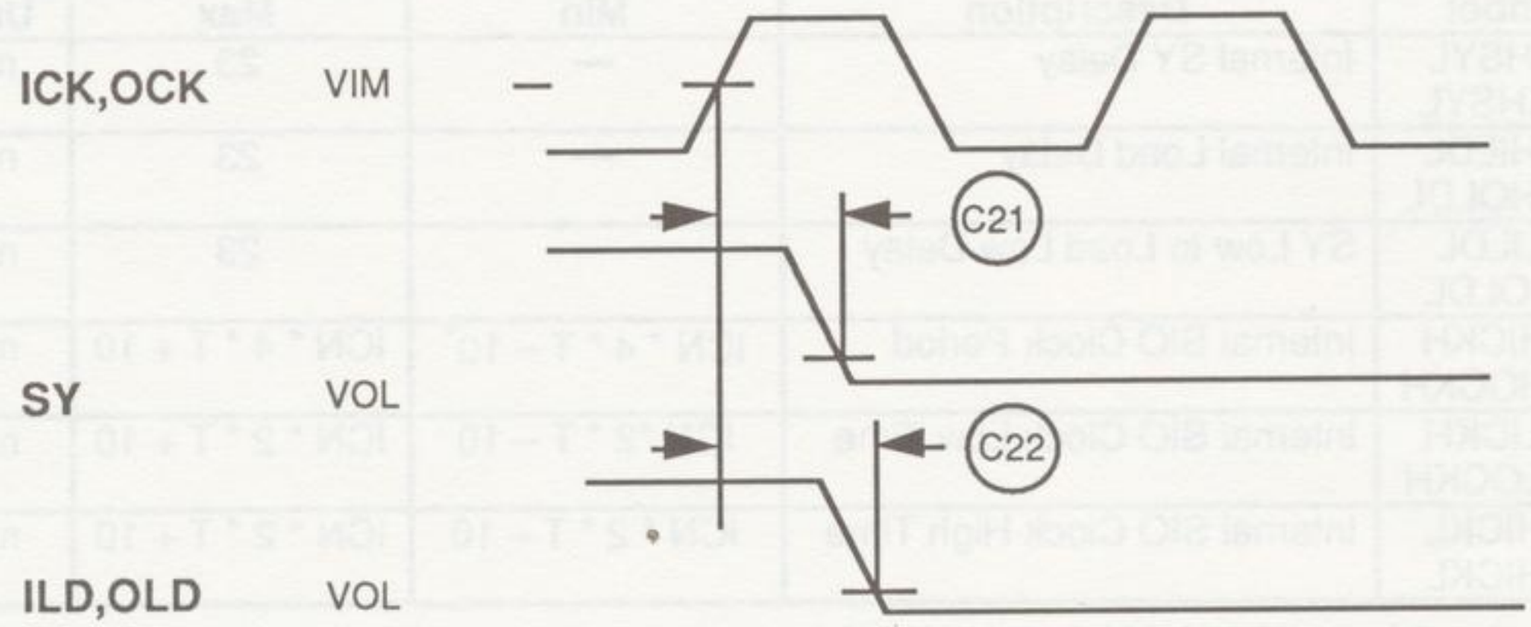
Table 12-12 Timing Characteristics for Serial Clock Generation (See Figure 12-7)

Num	Symbol	Description	Min	Max	Unit
C21	tICKHSYL tOCKHSYL	Internal SY Delay	—	23	ns
C22	tICKHILDL tOCKHOLDL	Internal Load Delay	—	23	ns
C23	tSYLILDL tSYLOLDL	SY Low to Load Low Delay	—	23	ns
C24	tICKHICKH tOCKHOCKH	Internal SIO Clock Period	ICN * 4 * T – 10*	ICN * 4 * T + 10	ns
C25	tICKLICKH tOCKLOCKH	Internal SIO Clock Low Time	ICN *2 * T – 10	ICN * 2 * T + 10	ns
C26	tICKHICKL tICKHICKL	Internal SIO Clock High Time	ICN * 2 * T – 10	ICN * 2 * T + 10	ns

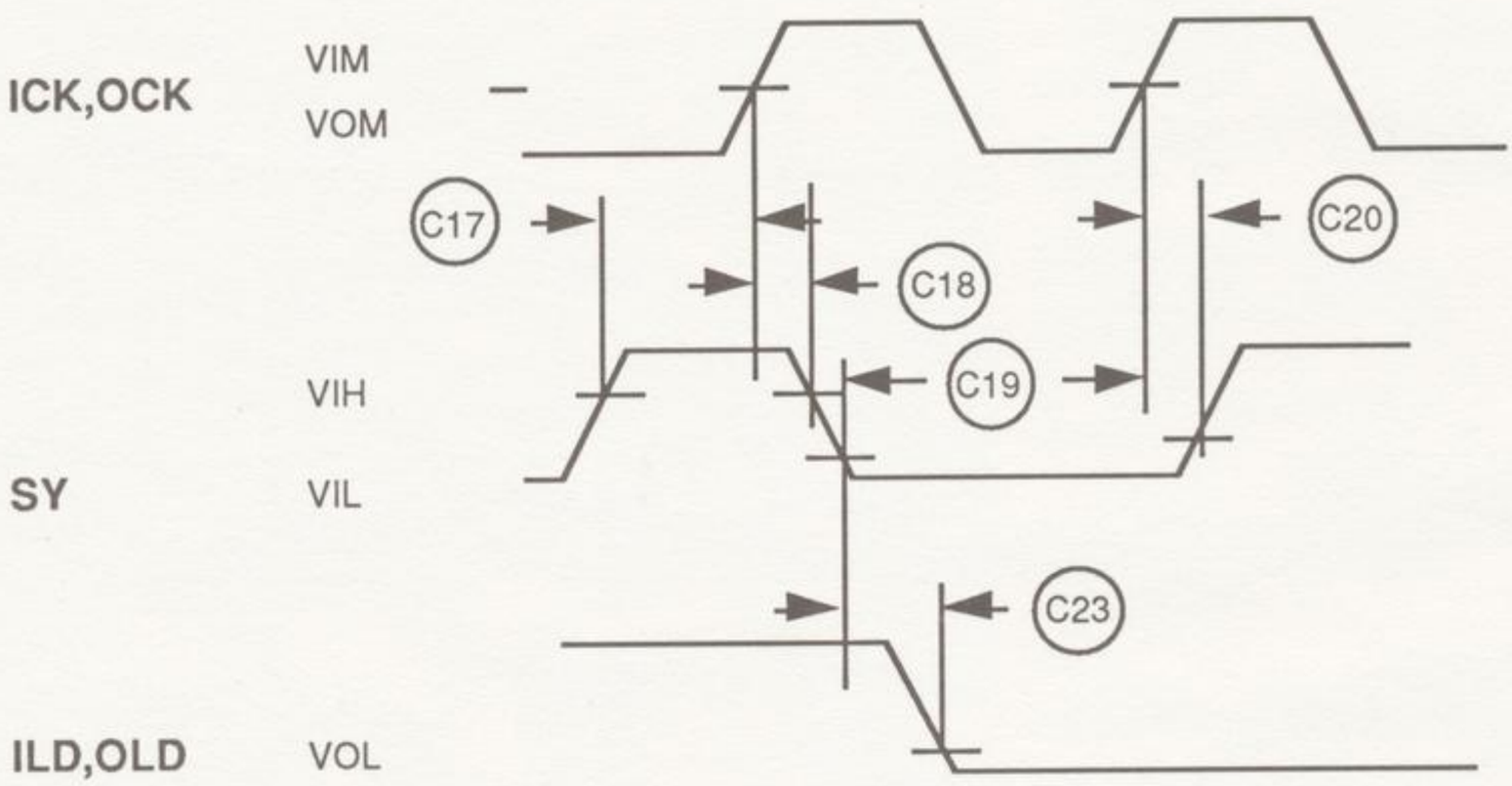
* ICN refers to the value of the four bit field IOC[23—20]. When ICN=0, the period of the clock is infinity. (The clock is disabled from counting.)



a. internal ICK/OCK



b. internal ILD/OLD/SY. external ICK/OCK



c. external SY. internal ILD/OLD
external or internal ICK/OCK

Figure 12-7 ac Serial Clock Generator Timing

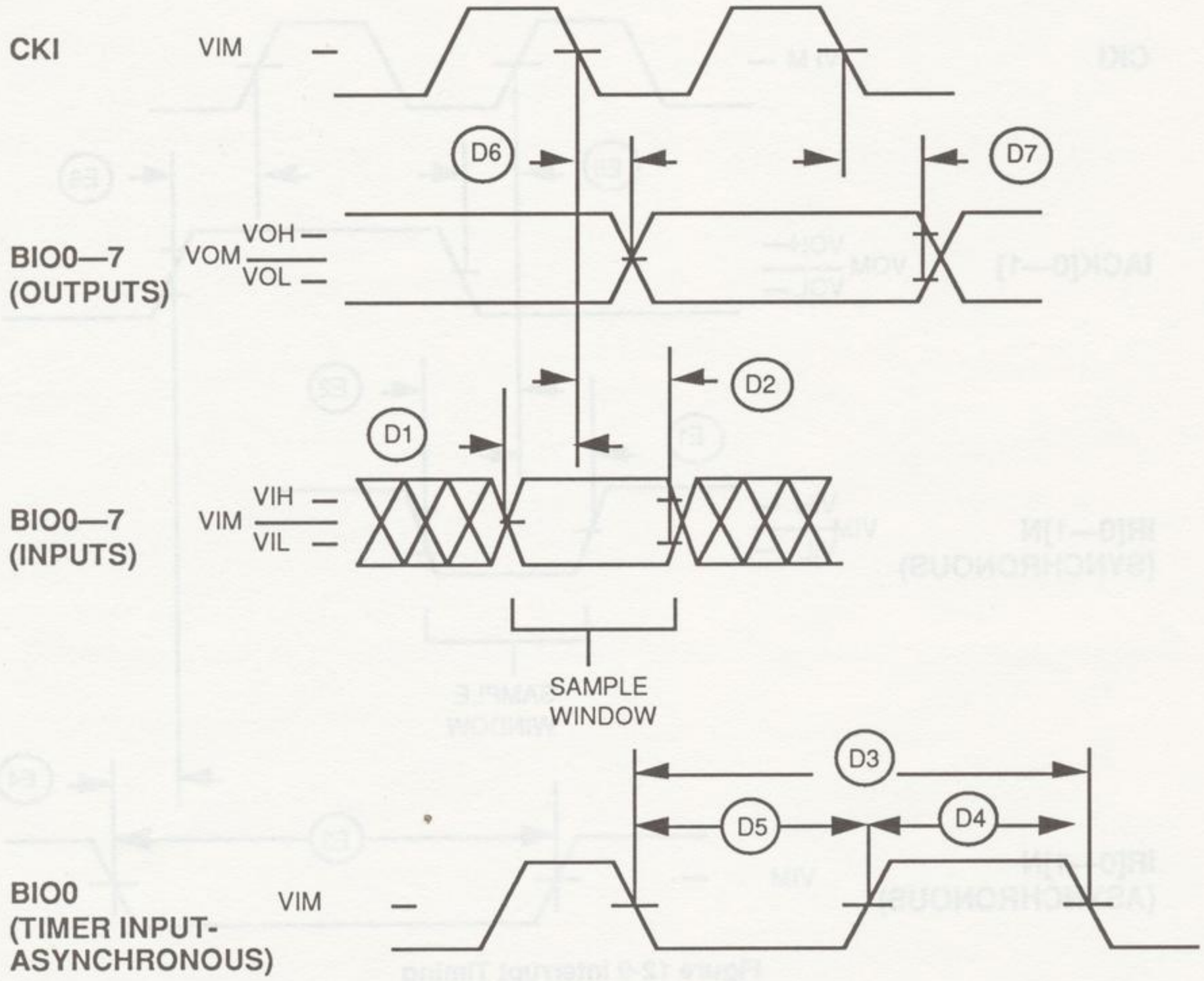


Figure 12-8 Bit I/O Timing

Table 12-13 Timing Requirements for Bit I/O (See Figure 12-8)

Num	Symbol	Description	Min	Max	Unit
D1	tBIOVCKIL	BIO Setup Time	4	—	ns
D2	tCKILBIOX	BIO Hold Time	2	—	ns
D3	tBIO0HBIO0H	Timer Clock Period	2T + 10	—	ns
D4	tBIO0HBIO0L	Timer Clock High Time	T + 5	—	ns
D5	tBIO0LBIO0H	Timer Clock Low Time	T + 5	—	ns

Table 12-14 Timing Characteristics for Bit I/O (See Figures 12-8)

Num	Symbol	Description	Min	Max	Unit
D6	tCKILBIOV	BIO Delay	—	16	ns
D7	tCKILBIOX	BIO Hold	4	—	ns

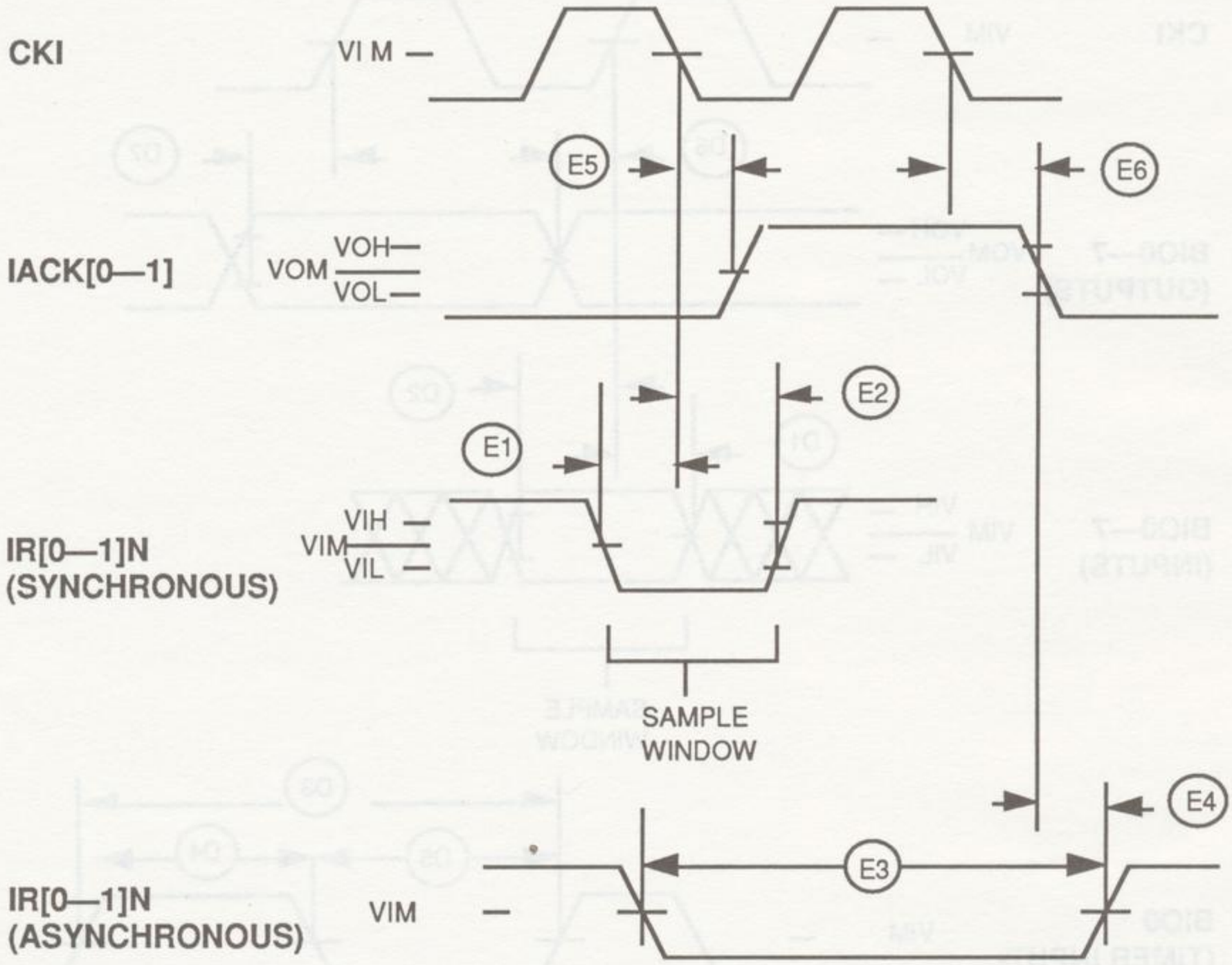


Figure 12-9 Interrupt Timing

Table 12-15 Timing Requirements for Interrupts (See Figure 12-9)

Num	Symbol	Description	Min	Max	Unit
E1	tIRLCKIL	Interrupt Asserted Setup Time (Synchronous)	4	—	ns
E2	tCKILIRX	Interrupt Asserted Hold Time (Synchronous)	2	—	ns
E3	tIRLIRH	Interrupt Asserted Pulse Width (Asynchronous)	T + 5	—	ns
E4	tIACKHIRH	Interrupt Acknowledge to Interrupt Negation	—	2T	ns

Table 12-16 Timing Characteristics for Interrupts (See Figure 12-9)

Num	Symbol	Description	Min	Max	Unit
E5	tCKILIACKL	Interrupt Acknowledge Delay	—	16	ns
E6	tCKILIACKX	Interrupt Acknowledge Hold	4	—	ns

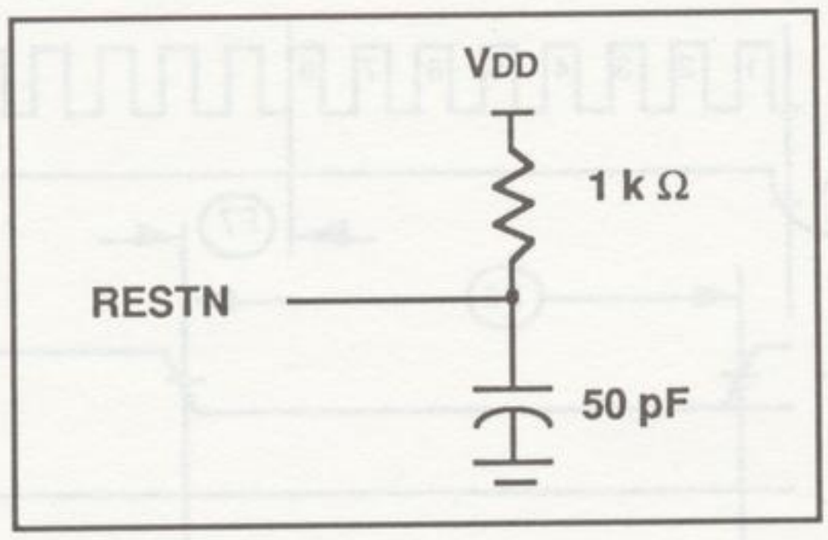


Figure 12-10 RESTN Test Load Circuit

Table 12-17 Timing Requirements for Reset (See Figures 12-10 to12-12)

Num	Symbol	Description	Min	Max	Unit
F1	tBIOVRESTH	BIO Setup to RESTN High*	8	—	ns
F2	tRESTHBIOX	BIO Hold After RESTN High	—	0	ns
F3	tRESTHZNH	ZN High After RESTN High	—	8T†	ns
F4	tRESTLRESTH	RESTN In Pulse Width	8T + 10	—	ns
F5	tRESTLRESTH	RESTN In Pulse Width	8T + 10	—	ns

* Only BIO[4:7] are latched into the device when RESTN makes a zero-to-one transition.
† ZN may be asserted at any time. This maximum specification permits correct DSP3210 operation in the run mode.

Table 12-18 Timing Characteristics for Reset and ZN (See Figures 12-10 to 12-12)

Num	Symbol	Description	Min	Max	Unit
F6	tRESTPU	Power-on RESTN Output Pulse Width	7T – 10	7T + 60	ns
F7	tCKILRESTV	Power-on RESTN Output Delay	—	60	ns
F8	tRESTLBIOZ	Power-on RESTN Low to BIO High-Z (Inputs)	—	9	ns
F9	tRESTLBISZ	Power-on RESTN Low to Bus Interface Signals High-Z*	—	9	ns
F10	tRESTHBISE	RESTN High to Bus Interface Signals Low-Z	—	12T + 12	ns
F11	tRESTLSOBZ	Power-on RESTN Low to Output and Biput Signals High-Z†	—	9	ns
F12	tRESTLSOBE	Power-on RESTN Low to Output and Biput Signals Low-Z	—	8T + 12	ns
F13	tZNLSOBZ	ZN Low to Output and Biput Signals High-Z	—	18	ns
F14	tZNHSOBE	ZN High to Output and Biput Signals Low-Z	—	18	ns
F15	tRESTILBIOZ	RESTN In Low to BIO High-Z (Inputs)	—	17	ns
F16	tRESTILBISZ	RESTN In Low to Bus Interface Signals High-Z	—	4T + 12	ns

* Bus Interface Signals are defined to be A2—A31, MS0—MS3, CSN, ASN, RW, PBD, MRN, MWN, and LOCKN.
† Outputs and Biputs are defined to be BRN, BGACKN, BIO[0—7], ICK, ILD, DO, OCK, OLD, SY, and IACK[0—1].

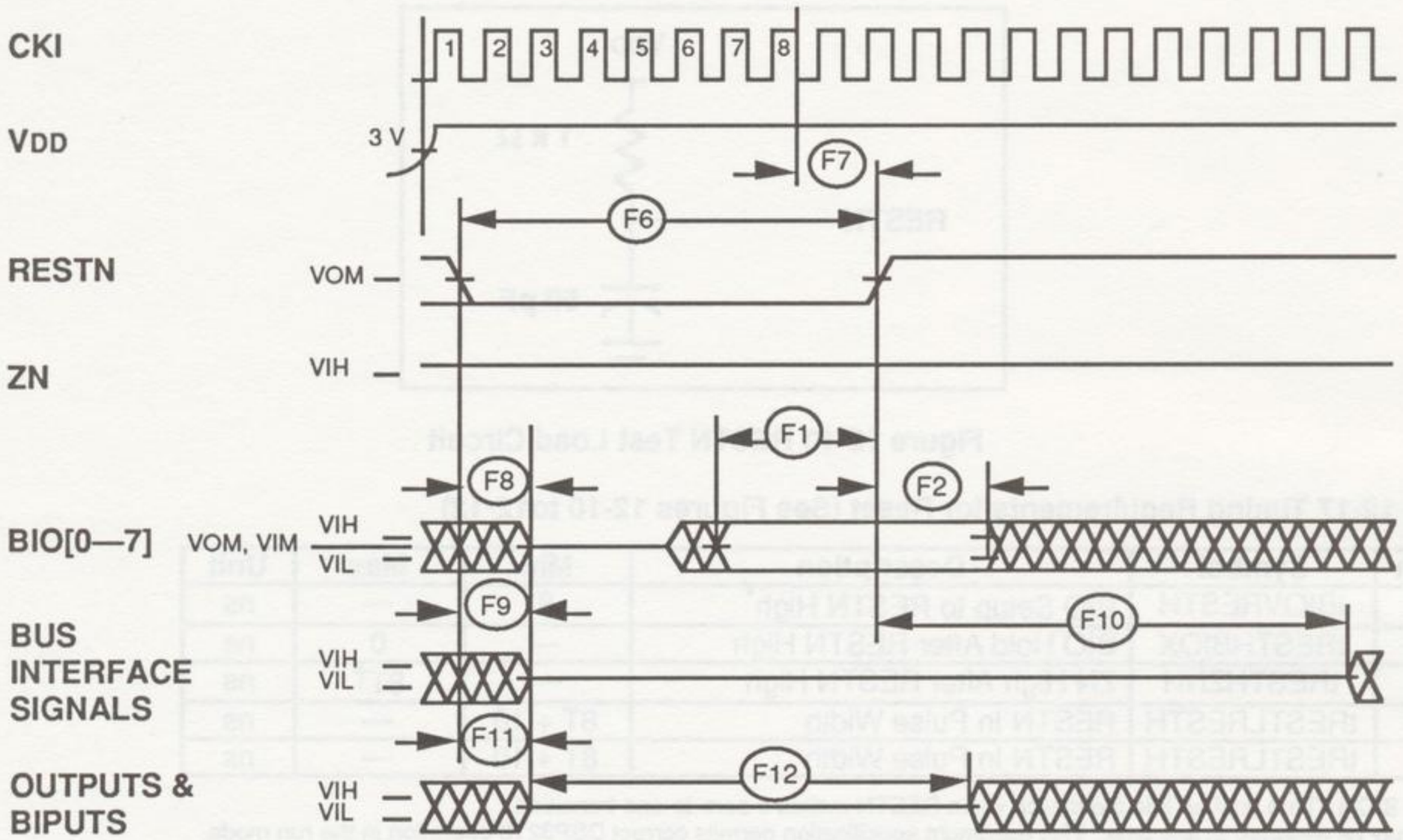


Figure 12-11 Power-on Reset Timing

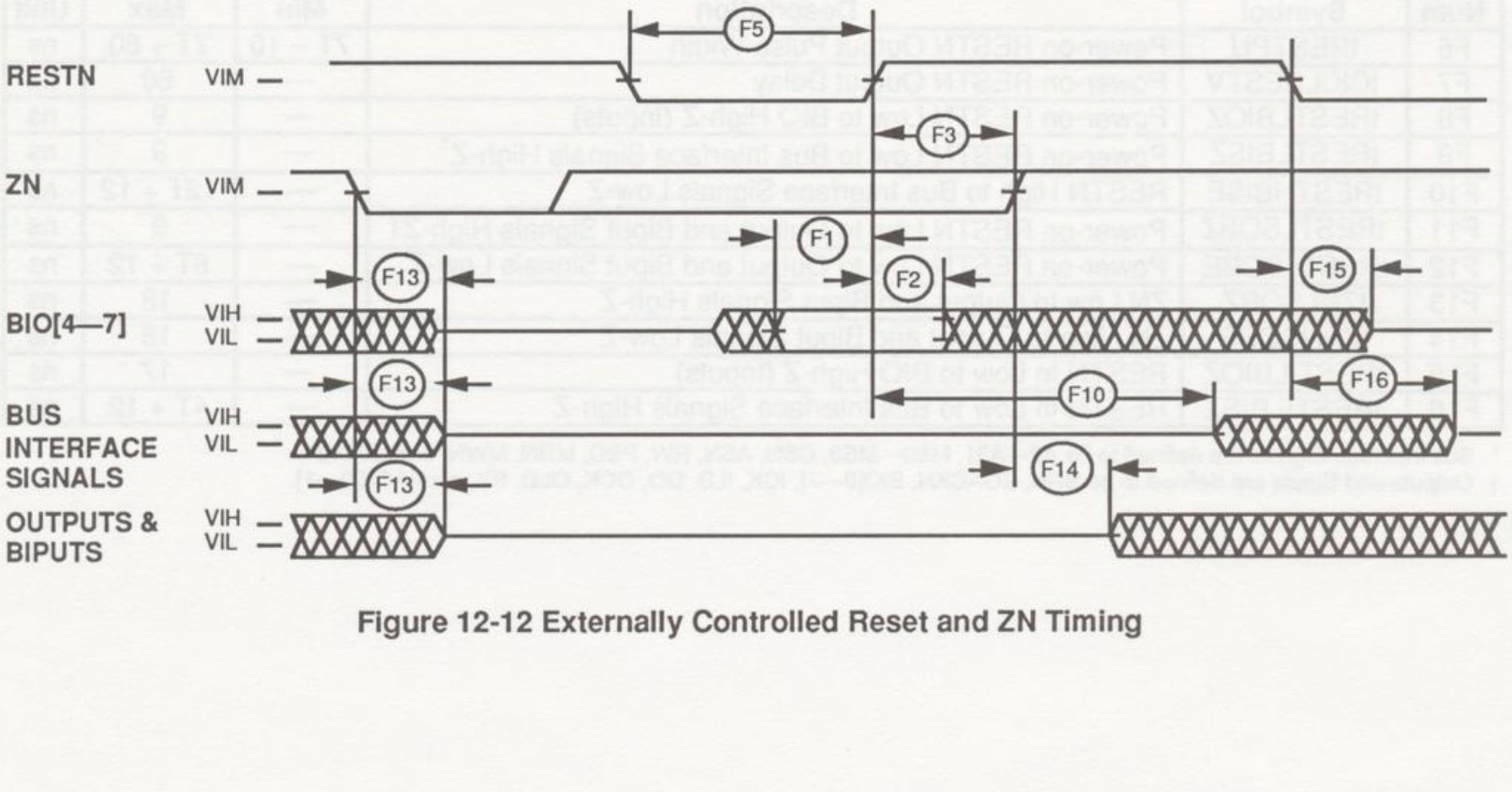


Figure 12-12 Externally Controlled Reset and ZN Timing

Appendix A

APPENDIX A

BOOT ROM CODE A-1
Table A-1 Operation of Boot ROM Routines A-1
Figure A-1 EPROM Data Organization A-2
Listing A-1 Boot ROM Code A-2

BOOT ROM CODE

Program execution always begins with the instruction at location 0x0. This physical address is either in Boot ROM or external memory depending on the the state of pcw[10], C/PN, when the DSP3210 is reset. Note that pcw[10—13] are initialized during the reset state with the logic states applied to bio[7—4], respectively. If C/PN is initialized to a one, execution begins from the first location in the Boot ROM. If C/PN is initialized to a zero, execution begins from location zero of external memory A.

When Computer mode is selected, the Boot ROM code is run on reset. The Boot ROM program uses pcw[13], BRC, to select between two routines stored in the ROM. Table A-1 lists a summary of the operations performed by each routine. The routine selected when pcw[13] = 0 is used to redirect the starting address of the DSP3210. The routine selected when pcw[13] = 1 is used to load a program from a external 8-bit wide memory, such as an EPROM, and execute it. The EPROM must be 8-bits wide and be mapped to external memory address 0x60000000. Figure A-1 shows the organization of data in the byte-wide EPROM when using little endian byte ordering.

A listing of the assembly code is provided in Listing A-1. Note that the fill code at the end of the ROM is for future expansion and should not be used.

Table A-1 Operation of Boot ROM Routines

pcw[13]	Operations
0	(1) Enables interrupt 1. (2) Enters wait-for-interrupt mode, and when interrupt 1 is acknowledged*. (3) Changes memory map to Processor mode. (4) Disables interrupt 1. (5) Reads location 0 (external memory) for branch vector and writes back a zero to location 0. (6) Returns from interrupt by loading pcsh with branch vector so that the DSP3210 starts executing from the branch vector.
1	(1) Read four bytes, organized as the least significant byte of the first four word addresses, from external memory starting at 0x60000000 into a 32-bit integer (define this as the count). (2) Read the next four bytes, organized as the least significant byte of the first four word addresses, from external memory starting at 0x60000010 into a 32-bit pointer (define this as the address). (2) If count is zero, branch to address. (3) If count is non-zero, read count number of bytes from external memory starting at 0x60000020 and write them to address to form words and then branch to address.

* For the code to operate correctly, the chip must enter the wait-for-interrupt mode before taking the interrupt. Note that the emr register enables interrupt 1 prior to the waiti instruction. If an interrupt is asserted between the enabling of interrupt 1 and the waiti instruction, the interrupt will be taken with the wrong memory map. The time from RESTN making a low-to-high transition to the program reaching the waiti instruction is 64 * T, where T is a CKI period.

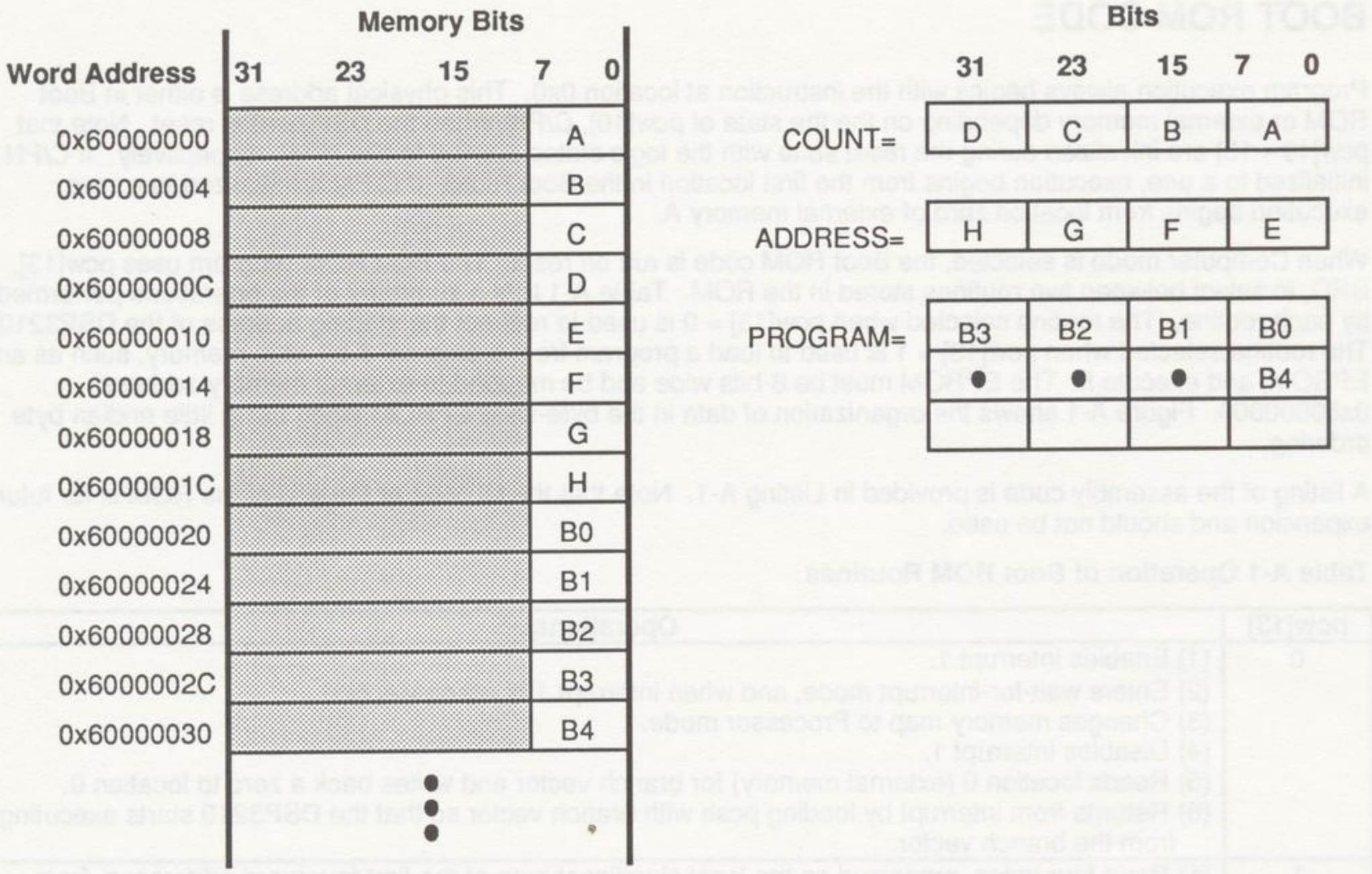


Figure A-1 EPROM Data Organization (Little Endian Byte Ordering)

Listing A-1 Boot ROM Code

```
/*
```

DSP3210 Boot ROM

This code is mask programmed into the BOOT ROM of the DSP3210.

SYNOPSIS:

The DSP3210 is reset via the RESTN pin and comes up in Computer mode by setting PCW bit 10 (C/PN) to 1 via the BIO7 pin. In addition, PCW bit 13 is set to either a 0 or 1 via the BIO4 pin. PCW bit 13 is used to determine which boot procedure to execute.

PCW Bit 13 = 0 Procedure:

The DSP3210 sets up r22 to point to 0x5003000 which is the address of the BOOT ROM in Processor mode. Next, the EMR register is set to enable External Interrupt 1. The DSP3210 then sets bit 10 of the PCW register to 0 without changing the other bits in PCW. Finally, the DSP3210 goes into Wait for Interrupt Mode with its memory map set for Processor mode.

When the system host decides to start the DSP3210, it places a vector to the DSP3210 external memory boot procedure into physical location zero. The system host then pulses IREQ1 pin (External Interrupt 1 Request) on the DSP3210 and begins polling physical location zero.

The DSP3210 acknowledges the interrupt via the IACK1 pin and vectors to location 0x50030078 which is in the BOOT ROM. The DSP3210 then clears the EMR register to prevent any further interrupts. Next, it loads the PCSH, pc shadow, register from location zero (which is now in external memory).

The DSP3210 then writes a zero into location zero. Finally, the DSP3210 performs a return from interrupt which executes a NOP from the Shadow Instruction register followed by the instruction which was fetched from the vector supplied by the host in physical location zero.

PCW Bit 13 = 1 Procedure:

The DSP3210 fetches 8 words starting from external memory 0x60000000. The least significant byte (LSB) of the first four words make up the address where the DSP3210 will branch to upon completing the boot procedure.

The LSB of the next four words contain the number of bytes which are to be copied from the location following these eight words into the memory range specified by the starting address.

Upon completion of the copy operation, the DSP3210 transfers program control to the starting address. Note that the memory map remains in the computer mode.

```
*/

/*
 *      I/O Registers Defines
 */

#define         ps      ior00
#define         emr     ior08
#define         spc     ior10
#define         pcw     ior12
#define         dauc    ior14

/*
 *      PSEUDO Instructions
 */

#define         waiti   spc = r0
#define         bkpt    spc = (short) r0
#define         reset   spc = (byte) r0

        .rsect  ".brom", TEXT

        /*
         *      3210 Exception Vector Table
         */

        pcgoto boot_begin       /* Reset */
        r1 = (short) pcw
        pcgoto boot_error       /* Bus Error */
        nop
        pcgoto boot_error       /* Illegal Opcode */
        nop
        pcgoto boot_error       /* Reserved */
        nop
        pcgoto boot_error       /* Address Error */
        nop
        pcgoto boot_error       /* DAU Overflow/Underflow */
        nop
        pcgoto boot_error       /* NAN */
        nop
        pcgoto boot_error       /* Reserved */
        nop
        pcgoto boot_error       /* External Interrupt 0 */
        nop
        pcgoto boot_error       /* Timer */
        nop
        pcgoto boot_error       /* Reserved */
```

```
        nop
        pcgoto boot_error       /* SIO Input Buffer */
        nop
        pcgoto boot_error       /* SIO Output Buffer */
        nop
        pcgoto boot_error       /* SIO DMA Input Frame */
        nop
        pcgoto boot_error       /* SIO DMA Output Frame */
        nop
        pcsh = (long) *r0       /* External Interrupt 1 */
        emr = (short) r0
        ireturn
        nop

boot_begin:
        r2 = r0 <<| 0x6000
        r1 & 0x2000
        if(eq) pcgoto system_boot
        r3 = (ushort24) 0xE000
        do 7 {
                r4 = *r2++
                nop
                *r3++ = (byte) r4
        }
        r5 = *0xE004    /* fetch number of bytes */
        r1 = *0xE000    /* fetch start address */
        if(eq) pcgoto boot_end  /* check number of bytes */
        r3 = r1
        r5 = r5 - 1
        do r5 {
                r4 = *r2++
                nop
                *r3++ = (byte) r4
        }
boot_end:
        goto r1 + 0
        nop

system_boot:
        r22 = r0 <<| 0x5003
        r2 = (ushort24) 0x8000
        emr = (short) r2
        r1 = r1 & (~0x0400)
        waiti
        pcw = r1
        *r0 = r0

boot_error:
        bkpt    /* signal error */
        pcgoto .
        nop

/* Fill remaining locations of the Boot ROM with their address */

        fltbits   (. - boot_rom + 0x50030000)
        fltbits   (. - boot_rom + 0x50030000)
        fltbits   (. - boot_rom + 0x50030000)
        .
        .
        .
```

Glossary

2's Complement Integer - A method used in some systems to represent positive and negative integers. Positive integers are identical to standard binary numbers; however, negative integers are the complement of a standard binary number plus one.

3-state - To place an output in the high-impedance state.

A-law - A European standard for the compression and expansion of the dynamic range of a signal.

Addressing Modes - Various common modes for addressing in the DSP3210 include immediate, memory direct, register direct, and register indirect.

ALU - Arithmetic logic unit

Arithmetic Logic Unit (ALU) - On-chip unit that performs arithmetic operations.

Assembler - A program that translates symbolically represented characters into a form (binary) the computer can interpret.

Bit I/O (BIO) - Input or output pins used for status and control.

Boot ROM - Preprogrammed Read-Only Memory that contains instructions that initialize operation of the DSP3210.

Bus Interface - A general-purpose parallel data port that can be used for interfacing to microprocessor buses, I/O buses, memory, or memory-mapped devices.

Byte - An 8-bit quantity that may appear at any address in memory.

CA - Control arithmetic.

CAU - Control arithmetic unit.

CISC - Complex Instruction Set Computer.

Companding - The process of compressing and expanding a signal.

Control Arithmetic (CA) Instructions - Instructions executed in the CAU that perform address generation, integer data arithmetic and logic functions, instruction flow control, and data movement.

Control Arithmetic Unit (CAU) - Execution unit that performs 16- or 32-bit, integer arithmetic for logic and control functions.

DA - Data arithmetic.

Data Arithmetic (DA) Instructions - 32-bit floating-point instructions that perform multiply/accumulate operations for signal-processing algorithms. DA instructions also contain special functions for performing data type and format conversions and rounding.

Data Arithmetic Unit (DAU) - Primary execution unit that performs 32-bit floating-point arithmetic for signal-processing functions.

Data Arithmetic Unit Control (DAUC) Register - A register that controls the type of conversions performed in the DAU on input and output data and controls rounding modes.

DAU - Data arithmetic unit.

DAUC - Data arithmetic unit control.

Direct Memory Access (DMA) - High-speed data transfer operation where an I/O channel transfers information directly to or from memory.

DMA - Direct memory access.

DMAC - Direct Memory Access Controller

EMR - Exception mask register.

Exception Vector Table - Sixteen pairs of 32-bit words that control the program flow based on the source of an exception.

Exception Vector Table Pointer (EVTP) - A register that locates the starting address of the exception vector table.

Fast Fourier Transform (FFT) - A complex procedure for transforming a time-domain waveform into its frequency-domain components.

FFT - Fast Fourier transform.

Floating-Point Arithmetic - A method of arithmetic in the computer that keeps track of the binary point.

Frame Boundary - The first time the output buffer is full following a high-to-low transition on the SY pin, provided IOC[15] is equal to zero at that time. Frame boundary is generated by the on-chip clock generator and is useful for certain TDM interfaces.

Halfword - A 16-bit quantity that may appear at any address in memory.

I/O - Input/output.

I/O Control (IOC) Register - A register used to configure the serial I/O interface to external devices. It is used to set various I/O configurations, bit lengths, internal or external clock, and internal or external synchronization.

IBUF - Input buffer.

Immediate Mode - An addressing mode in which the operand is supplied by the instruction.

Indexed Mode - An addressing mode in which the address part of the instruction is modified by an auxiliary (index) register during the execution of that instruction. This mode is achieved with the instruction *rD = rS + N* followed by **rD*.

Input Buffer (IBUF) - A register used to accept serial input from an external device.

Input Shift Register (ISR) - An input register for converting serial input data to parallel data.

Interrupt - A means by which external devices may request service by the microprocessor.

IOC - Input/output control.

ISR - Input shift register.

Link Editor - Software support tool used for maintaining libraries, defining sections of on-chip and off-chip memory, and recognizing holes in memory.

Memory Direct Mode - An addressing mode in which the instruction contains the address of an operand in memory.

Memory Indirect Mode - An addressing mode in which a memory location contains the address of the data, rather than the data itself. This mode is achieved with the instruction *rD = MEMORY* followed by **rD*.

Memory Mapped I/O (MMIO) - Instruction referencing method for on-chip peripherals.

Multimedia - The processing of multiple forms of data, including sound, speech, data, images, and video.

OBUF - Output buffer.

OSR - Output shift register.

Output Buffer (OBUF) - A register used to pass data to the output shift register.

Output Shift Register (OSR) - A register for converting parallel data from the output buffer into a serial data stream.

PCW - Processor control word.

Pipelining - Overlapping the execution of instructions to increase performance.

Processor Status (PS) Register - Register that displays flags for the CAU, DAU, and I/O.

Register Direct Mode - An addressing mode in which the instruction specifies a register that contains the operand.

Register Indirect Mode - An addressing mode in which the instruction contains the name of a pointer register (rP) that contains the address of the operand.

Register Mapped I/O (IO) - Instruction referencing method for DSP3210 core control registers.

Relative Mode - An addressing mode in which the absolute address is obtained by means of address modification. Address modification is performed by the addition of a given number to the address part of an instruction known as the relative address. This mode is attained by moving the program counter (pc) to another CAU register and then performing a register indirect operation. In relative addressing, the address contained in the second byte of the instruction is added to the program counter's lowest eight bits plus one.

RISC - Reduced Instruction Set Computer.

Serial I/O (SIO) - A group of registers that allows interfacing with other devices using few, if any, external chips. It converts serial input data and parallel output data into serial data.

Simulator - A highly specific program that allows accurate simulations of the logical operation of the DSP3210 device.

SIO - Serial input/output.

Stack - An area of reserved memory used for storing the program counter and the contents of registers during a program interrupt.

Time-Division Multiplexed (TDM) - A procedure for transmitting two or more signals over a common channel through the use of successive time intervals for different devices.

Timer - Programmable counter.

TSC - Timer/Status and Control

Word - A 32-bit quantity that may appear at any address in memory.

µ-law - A United States standard for compressing and expanding the dynamic range of a signal.

µP - Microprocessor.

INDEX

INDEX

μ-law 4-5,50,63, 8-13

— A —

24-bit immediate 4-71
8-bit unsigned conversions 4-50,63
A-law 4-5,50,63, 8-13
adaptive filter 4-17
add 4-21,22,23, 8-3
add, carry reverse 4-24, 8-3
Addressing modes
  24-bit immediate 3-12, 4-11
  32-bit immediate 3-13
  bit-reversed 3-13
  memory direct 3-12
  modulo 3-14
  overview 3-11
  register direct 3-12
  register indirect 3-13
  short immediate 3-12, 4-11
AND 4-25,27, 8-3
AND with complement 4-26
application library 1-7, 2-1
applications 2-1

— B —

base level 7-3
BIO
  block diagram 9-26
  overview 3-5, 9-24
  pins 5-1
  programming model 9-25
  timing 9-26
bio register 10-12, 9-25
bioc register 10-12, 9-25
bit-reversal 4-18
Boot ROM 3-3,9, 7-2
Bulletin Board 1-7
Bus interface
  asynchronous 6-18
  bus arbitration 3-6, 6-12
  bus error 6-8,11
  interlocked access 6-15
  multiplexed bus 6-17
  overview 3-4
  page break detect 6-15
  pins 5-1
  quad-word block move 6-16
  read transaction 6-7
  relinquish/retry 6-8
  retry 6-8,11
  signals 6-1
  standard cycles 6-6
  write data duplication 6-2
  write transaction 6-7
byte 3-9, 4-10
byte addressable 3-8
byte ordering 3-8

— C —

C Language Compiler 1-7
CAU
  block diagram 8-4
  evtp 8-1
  flags 4-3
  functions 8-3
  overview 3-3
  performance 3-3
  programming model 8-1
  ps 8-2
  ps register 8-1
  r0 register 8-1
  registers 4-1, 8-1
  stack pointer 8-1
  Z-address delay 8-5
codecs 3-4
ctr register 10-7
cycle stealing 3-5, 7-2

— D —

data types 3-6
DAU
  accumulators 8-7
  arithmetic precision 8-10
  block diagram 8-9
  ctr register 8-7, 8
  data type conversions 3-3, 8-7,12,13
  dauc register 8-7, 8
  flags 4-3
  floating-point format 3-3, 8-8
  overview 3-3
  performance 3-3
  programming model 8-7
  registers 4-1
  rounding modes 8-14
dauc register 10-6
decrement 4-55, 8-3
divide 4-18
divide seed 4-5,69, 8-14
DMA 3-4, 5, 7-2, 9-2
DMAC
  block diagram 9-19
  DMA cycle 7-2
  MMIO 3-10
  overview 3-4, 9-18

programming model 9-18
registers 9-18
dmac register 10-10, 9-20
do loop 4-6,31,32
doblock 4-6,31,32
documentation 1-7
dolock 4-31,32
double error level 7-3
DSP32 overview 1-4
DSP3210
block diagram 3-2
features 3-1
key attributes 3-1
overview 1-5
programming model 4-2
registers 4-1
DSP32C overview 1-4

— E —

Electrical Specifications
AC parametrics 12-2
DC parametrics 12-1
emr 3-4
emr register 10-7, 3-4,5, 7-4,6
endian 3-9
error level 7-3
error trace register 7-8, 8-2
evtp register 7-4
Exceptions
abort model 7-3,8
address error 7-10
bus error 7-10
continuation model 7-3,5
DAU underflow/overflow 7-10
double error 7-9
errors 3-5
errors 7-4,8
IEEE NAN 7-10
illegal opcode 7-10
interrupts 3-5, 7-4
levels 7-5
maskable 7-6
non-maskable 7-6
overview 3-5
priority 7-5
quick interrupt 7-8
vector table 7-4,5,8
Exclusive OR 4-34, 8-3
external partitions 3-4, 9, 7-2

— F —

FFT butterfly 4-18
FIR filter 4-17
flags 4-3, 8-2
Floating-point data
format 3-7, 8-10
IEEE format 8-12
implicit bit 8-11
range 3-8
zero 3-8

— G —

graphics 2-3, 8-7

— H —

half-word 3-9, 4-10

— I —

I/O flags 4-3
ibuf register 3-4
IEEE conversions 4-5, 33,51
IIR filter 4-17
increment 4-55, 8-3
instruction cycle 7-1
instruction set 4-1
Instructions
arithmetic/logic 4-8, 9
CA 4-1, 6
control 4-6
DA 4-1, 4
data move 4-10
doblock 6-16
dolock 6-15
encodings 10-1
IO register load 4-61
IO register store 4-82
latency 4-14, 15
multiply/accumulate 4-4
not interruptible 7-7
register load 4-59, 60
register store 4-80, 81
restrictions 4-12, 13
special functions 4-5
tap delay 4-41
integer conversions 4-5,38,56,57,
Integers
16-bit 3-7
32-bit 3-6
interlock 7-4
interrupt level 7-3
interrupt shadow 3-5
IO registers 3-14, 4-1,10
ioc register 3-4, 9-4, 10-9

— M —

memory map 3-10
MMIO 3-4,5,9, 9-1

MMIO map 3-11
mu-law (See μ-law)
multimedia 2-2

— O —

obuf register 3-4
OR 4-64, 8-3

— P —

package,132-pin PQFP 11-1
pcw 6-3
pcw register 3-9, 10-7,8
peripherals 3-10
product support 1-7
ps register 10-6

— R —

RAM 3-3,9, 7-2
Register encodings 10-6
Reset 3-9, 7-11
Reset, power-on 7-11,12
Reset, soft 4-6,72
Reset, timing 7-12
rotate, left 4-66, 8-3
rotate, right 4-67, 8-3

— S —

sales offices 1-8
shift and OR 4-75, 8-3
shift,arithmetic right 4-78,79, 8-3
shift, left 4-73,74, 8-3
shift, right 4-76,77, 8-3
short immediate 4-70
signal processing functions 4-17
Signals
  3-state 5-4
  A2-A31 5-2, 6-1
  active state 5-4
  AEN 6-17
  AEN/MRN 5-2, 6-3
  ASN 5-2, 6-1
  BERRN 5-3, 6-3
  BGACKN 5-3, 6-12
  BGN 5-3, 6-12
  BIO0-BIO7 5-3
  BLMN 6-2, 16
  BRN 5-3, 6-12
  CKI 5-3
  CSN 5-2, 6-1
  D0-D31 5-2, 6-1
  DEN 6-17
  DEN/MWN 5-2, 6-3
  DI 5-3, 9-5
  DLE 5-2, 6-3,18
  DO 5-3, 9-5
  IACK0 5-3
  IACK1 5-3
  ICK 5-3, 9-5
  ILD 5-3, 9-5
  IR0N 5-3
  IR1N 5-3
  LOCKN 5-3, 6-2,15
  MS0-MS3 5-2, 6-1
  OCK 5-3, 9-5
  OLD 5-3, 9-5
  PBD 6-2
  PBD/BLMN 5-2
  pin assignments 11-2
  RESTN 5-3, 7-12
  RW 5-2, 6-1
  SRDYN 5-3, 6-3
  SY 5-3, 9-5
  ZN 5-3
SIO
  block diagram 9-3
  clock generation 9-12
  clock generator timing 9-12
  DMA 9-17
  frame boundary 9-14
  ibuf 9-1
  input port 9-7
  input protocols 9-8
  input timing 9-7
  interrupts 9-16
  ioc 9-1
  loss of sanity 9-13
  loss of sync 9-13
  MMIO 3-10
  obuf 9-1
  output port 9-9
  output protocol 9-10
  output timing 9-10
  overview 3-4
  pins 5-1
  programming model 9-1
  SIP 9-5
  SOP 9-9
software tools 1-6
states 7-1
subtract 4-30,83,84, 8-3

— T —

tcon register 9-23, 10-11
TDM 3-4
timer
  block diagram 9-22
  overview 3-5, 9-21
  programming model 9-21
  registers 9-21

timing 9-24
TSC 3-10

— W —

wait for interrupt 3-6, 4-6, 4-85, 7-3
wait-states 3-4, 9, 6-5, 7-2
wait-states, conflict 7-2
wait-states, external 7-2
waiti 4-85
word 3-8, 4-10

CA Arithmetic/Logic Instructions		
Instruction*	CAU Flag	Instr. Ref.
rD = [(short)] rS3+N	nzvc	ADD
rD = (long) rS <<| N	nz00	SHIFT-OR
rD = [(short)] rD - N	nzvc	SUBTRACT
rD = [(short)] N - rD	nzvc	SUBTRACT
rD = [(short)] rD & N	nz00	ADD
rD = [(short)] rD | N	nz00	OR
rD = [(short)] rD ^ N	nz00	EXCLUSIVE OR
rD = [(short)] rD # N	nz0c	ADD-CARRY REV
rD = [(short)] rD << N	nz00	SHIFT LEFT
rD = [(short)] rD >> N	nz00	SHIFT RIGHT
rD = [(short)] rD $>> N	nz00	SHIFT RIGHT-ARITH
[(short)] rD - N	nzvc	COMPARE
[(short)] rD & N	nz00	BIT TEST
[if(COND)] rD = [(short)] rS1 + rS2	nzvc	ADD
[if(COND)] rD = [(short)] rS1 - rS2	nzvc	SUBTRACT
[if(COND)] rD = [(short)] rS1 & rS2	nz00	ADD
[if(COND)] rD = [(short)] rS1 &~ rS2	nz00	AND-COMPLEMENT
[if(COND)] rD = [(short)] rS1 | rS2	nz00	OR
[if(COND)] rD = [(short)] rS1 ^ rS2	nz00	EXCLUSIVE OR
[if(COND)] rD = [(short)] rS1 # rS2	nz0c	ADD-CARRY REV
[if(COND)] rD = [(short)] rS1 << rS2	nz00	SHIFT LEFT
[if(COND)] rD = [(short)] rS1 >> rS2	nz00	SHIFT RIGHT
[if(COND)] rD = [(short)] rS1 $>> rS2	nz00	SHIFT RIGHT-ARITH
[if(COND)] [(short)] rS1 - rS2	nzvc	COMPARE
[if(COND)] [(short)] rS1 & rS2	nz00	BIT TEST
[if(COND)] rD = [(short)] rS/2	nz00	SHIFT RIGHT-ARITH
[if(COND)] rD = [(short)] rS>>1	nz00	SHIFT RIGHT
[if(COND)] rD = [(short)] rS<<1	nz00	SHIFT LEFT
[if(COND)] rD = [(short)] rS*2	nzvc	ADD
[if(COND)] rD = [(short)] rS>>>1	nz0c	ROTATE RIGHT
[if(COND)] rD = [(short)] rS<<<1	nzvc	ROTATE LEFT
[if(COND)] rD = [(short)] -rS	nzvc	SUBTRACT
[if(COND)] rD = [(short)] rS	nzvc	ADD
[if(COND)] rD1 = [(short)] rS {+, -}1	nzvc	INCR/DECR
[if(COND)] sp = (long) sp {++, --}	nzvc	INCR/DECR

* The size of arithmetic/logic and data move operations can be specified to be long (32-bits), or short (16-bits). Long is the default and does not have to be specified.

AT&T Microelectronics

AT&T DSP3210 Programmer's Reference Card

Note: { } and [ ] are not part of the instruction syntax. Parentheses ( ) are part of the syntax and must appear where shown in an instruction. Lower-case letters are part of the syntax and upper-case letters are replaced by immediate data, register names, or expressions.

DA Multiply/Accumulate Instructions		
Instruction	DAU Flag	Instr. Reference
[Z=] aN = [-]aM {+,-}Y*X	NZVU	FMULT-ACC-STORE
aN = [-]aM {+,-}(Z=Y)*X	NZVU	FMULT-ACC-TAP
[Z=] aN = [-]Y {+,-}aM*X	NZVU	FMULT-ADD-STORE
[Z=] aN = [-]Y *X	NZVU	FMULT-STORE
aN = [-] (Z=Y)*X	NZVU	FMULT-TAP
[Z=] aN = [-]Y {+,-} X	NZVU	FMULT-STORE
[Z=] aN = [-]Y	NZVU	FLOAD-STORE
aN = [-] (Z=Y) {+,-} X	NZVU	FADD-TAP

Replacement for DA Multiply/Accumulate Instructions	
Replace	Value
aN, aM	a0-a3
X, Y	*{r1—r14}, *{r1—r14}++, *{r1—r14}--, *{r1—r14}++{r15—r19}, a0-a3
Z	*{r1—r14}, *{r1—r14}++, *{r1—r14}--, *{r1—r14}++{r15—r19}

DA Special Function Instructions		
Instruction	DAU Flag	Instr. Reference
[Z=] aN = ic(Y)	NZ00	IC
[Z=] aN = oc(Y)	—	OC
[Z=] aN = float16(Y)	NZ00	FLOAT16
[Z=] aN = float32(Y)	NZ00	FLOAT32
[Z=] aN = int16(Y)	—	INT16
[Z=] aN = int32(Y)	—	INT32
[Z=] aN = round(Y)	NZVU	ROUND
[Z=] aN = ifalt(Y)	—	IFALT
[Z=] aN = ifaeq(Y)	—	IFAEQ
[Z=] aN = ifagt(Y)	—	IFAGT
[Z=] aN = dsp(Y)	NZVU	DSP
[Z=] aN = ieee(Y)	—	IEEE
[Z=] aN = seed(Y)	NZ0U	SEED

Restriction 1 - Z-pointer may only be used as Z-pointer in the following instruction.
Restriction 2 - CAU or IO register store cannot follow a DA instruction with a Y field memory reference.
Restriction 3 - CAU register loads cannot be referenced in the following instruction.

Replacement for DA Special Function Instructions	
Replace	**Value**
aN	a0-a3
Y	*{r1—r14}, *{r1—r14}++, *{r1—r14}--, *{r1—r14}++{r15—r19}, a0-a3 (Y may not be a0—a3 for *dsp* function.)
Z	*{r1—r14}, *{r1—r14}++, *{r1—r14}--, *{r1—r14}++{r15—r19}

CA Control Instructions		
Instruction	**CAU Flag**	**Instr. Reference**
if (COND) goto {N, rB, rB+N}*	—	GOTO-COND
if (rM-->=0) goto {N, rB, rB+N}*	—	GOTO-LOOP
goto {N, rB, rB+N, M, rB+M}*	—	GOTO
nop	—	NOP
call {N, rB, rB+N, M} (rM)*	—	CALL
return (rM)	—	RETURN
{do,dolock} K, {L, rM}	—	DO
doblock {L, rM}	—	DO
ireturn	—	IRETURN
sftrst	—	SFTRST
waiti	—	WAITI

*When using pc as rB, goto and call instructions may be written with the syntax *pcgoto label* or *pccall label*, respectively. The linker computes the appropriate N value to encode.

Replacement for CA Control Instructions	
Replace	**Value**
rB	pc, r0—r22
rM	r1—r22
N	16-bit signed integer
M	24-bit unsigned integer
COND	One of the DSP3210 Condition Codes

(COND) - Conditions					
CAU	**Flag**	**DAU**	**Flag**	**IO**	**Flag**
false	NA	ane	Z=0	ibe	ibf=0
true	NA	aeq	Z=1	ibf	ibf=1
pl	n=0	age	N=0	obe	obe=1
mi	n=1	alt	N=1	obf	obe=0
ne	z=0	avc	V=0	syc	sy=0
eq	z=1	avs	V=1	sys	sy=1
vc	v=0	auc	U=0	fbc	fb=0
vs	v=1	aus	U=1	fbs	fb=1
cc	c=0	agt	N | Z=0	ir0s	ir0n=1
cs	c=1	ale	N | Z=1	ir0c	ir0n=0
ge	n^v=0			ir1s	ir1n=1
lt	n^v=1			ir1c	ir1n=0
gt	z | (n^v)=0				
le	z | (n^v)=1				
hi	c | z=0				
ls	c | z=1				

Replacement for CA Arithmetic/Logic Instructions	
Replace	**Value**
rD	r0—r22
rD1	r0—r20, r22
rS, rS1, rS2	r0—r22
rS3	r0—r22, pc, pcsh
N	16-bit signed integer
COND	One of the DSP3210 Condition Codes

CA Data Move Group Instructions		
Instruction**	**CAU Flag**	**Instr. Ref.**
rD= (short) N	nz00	SET
rD= (ushort24) M	—	SET24
{MEM, *L, iorD} = {(byte),(hbyte),(short),(long)} rS	—	STORE
rD = {(char),(byte),(hbyte)} {MEM,*L}	nz00	LOAD
rD = {(short),(ushort)} {MEM,*L}	nz00	LOAD
rD = (long) {MEM, *L}	nz00	LOAD
rD = {(char),(byte),(hbyte)} iorS	nz00	LOAD
rD = {(short),(ushort)} iorS	nz00	LOAD
rD = (long) iorS	nz00	LOAD
MEM = {(byte),(short),(long)} iorS	—	STORE -IOR
iorD = {(byte),(short),(long)} MEM	—	LOAD-IOR

** The size of data move operations can be specified to be long (32-bits), short (16-bits), or byte (8-bits). Long is the default and does not have to be specified.

Replacement for CA Data Move Instructions	
Replace	**Value**
rS	r0—r22, pc, pcsh
rD	r0—r22
MEM	*{r0—r22}, *{r0—r22}++, *{r0—r22}--, *{r0—r22}++{r0—r22}
iorS, iorD	ior{0, 8, 10, 12, 14, 15}
N	16-bit number
L	16-bit number (unsigned)
M	24-bit number (unsigned)

Latency 1 - DA instruction writes may not be referenced until four instructions later.
Latency 2 - An accumulator may not be used as a multiplier input until three instructions later.
Latency 3 - goto, call, and return instructions are delayed by one instruction.
Latency 4 - DAU conditions are established no sooner than four instructions prior to the test.

For additional information, contact your AT&T Account Manager or the following:

❑ AT&T Microelectronics
Dept. 52AL040420
555 Union Boulevard
Allentown, PA 18103
1-800-372-2447
FAX 215-778-4106

In Canada, call:
1-800-553-2448
FAX 215-778-4106

❑ AT&T Microelectronics
AT&T Deutschland GmbH
Bahnhofstr. 27A
D-8043 Unterfoehring
Germany
Tel. 089/950 86-0
Telefax 089/950 86-111

❑ AT&T Microelectronics Asia/Pacific
14 Science Park Drive
#03-02A/04 The Maxwell
Singapore 0511
Tel. (65) 778-8833
FAX (65) 777-7495
Telex RS 42898 ATTM

❑ AT&T Microelectronics
AT&T Japan Ltd.
31-11, Yoyogi 1-chome
Shibuya-ku, Tokyo 151
Japan
Tel. (03) 5371-2700
FAX (03) 5371-3556

❑ AT&T Microelectronica de España
Poligono Industrial de
Tres Cantos (Zona Oeste)
28770 Colmenar Viejo
Madrid, Spain
Tel. (34) 1-8071441
FAX (34) 1-8071420

AT&T reserves the right to make changes to the product(s) or circuit(s) described herin without notice. No liability is assumed as a result of their use or application. No rights under any patent accompany the sale of any such product or circuit.

Copyright © 1991 AT&T
All Rights Reserved
Printed in U.S.A.

September 1991
MN91-006DMOS

AT&T
Microelectronics