microelectronics group

Lucent Technologies
Bell Labs Innovations

DSP16210 Digital Signal Processor

Features

- Optimized for applications requiring large internal memory, flexible I/O, and high cycle efficiency speech coding, speech compression, and channel coding
  - Large on-chip dual-port RAM (60 Kwords of DPRAM)—eliminates need for fast external SRAM
  - 2-input 40-bit arithmetic logic unit (ALU) with add/compare/select (ACS) for Viterbi acceleration
  - 3-input adder
  - DMA-based I/O—minimizes DSP core overhead for I/O processing
  - Flexible power management modes for low system power dissipation
  - Provides 200 DSP MIPS
- 10 ns instruction cycle time at 3 V
- Dual 16 x 16-bit multiplication and 40-bit accumulation in one instruction cycle for efficient algorithm implementations
- 31-instruction by 32-bit interruptible do-loop cache for high-speed, program-efficient, zero-overhead looping
- Nested interrupts and three interrupt priority levels for efficient control and task management operations
- On-chip boot ROM with hardware development system and boot code for flexible downloading
- On-chip, programmable, PLL clock synthesizer
- Enhanced serial I/O (ESIO) port designed to multiplex/demultiplex 64 Kbits/s, 32 Kbits/s, 16 Kbits/s, and 8 Kbits/s channels
- 26 Mbits/s simple serial I/O (SSIO) port coupled with DMA to support low-overhead I/O
- 16-bit parallel host interface (PHIF16) coupled with DMA to support low-overhead I/O
  - Supports either 8-bit or 16-bit external bus configurations (8-bit external configuration supports either 8-bit or 16-bit logical transfers)
  - Supports either *Motorola*[1] or *Intel*[2] protocols
- 8-bit control I/O interface for increased flexibility and lower system costs
- *IEEE*[3] 1149.1 test port (JTAG boundary scan)
- Full-speed in-circuit emulation hardware development system on-chip with eight address and two data watchpoint units for efficient application development
- Pin compatible with the DSP1620
- 144-pin TQFP package

Description

The DSP16210 is the first DSP device based on the DSP16000 digital signal processing core. It is manufactured in a 0.35 μm CMOS technology and offers a 10 ns instruction cycle time at 3 V operation. Designed specifically for applications requiring a large amount of memory, a flexible DMA-based I/O structure, and high cycle efficiency, the DSP16210 is a signal coding device that can be programmed to perform a wide variety of fixed-point signal processing functions. The DSP16210 includes a mix of peripherals specifically intended to support processing-intensive but cost-sensitive applications.

The large on-chip RAM (60 Kwords of dual-port RAM) supports downloadable system design—a must for infrastructure applications—to support field upgrades for evolving coding standards. The DSP16210 can address up to 192 Kwords of external storage in both its code/coefficient memory address space and data memory address space. In addition, there is an internal boot ROM (IROM) that includes system boot code and hardware development system (HDS) code.

This device also contains a bit manipulation unit (BMU) and a two-input, 40-bit arithmetic logic unit (ALU) with add/compare/select (ACS) for enhanced signal coding efficiency and Viterbi acceleration.

To optimize I/O throughput and reduce the I/O service routine burden on the DSP core, the DSP16210 is equipped with two modular I/O units (MIOUs) that manage the simple serial I/O port (SSIO) and the 16-bit parallel host interface (PHIF16) peripherals. The MIOUs provide transparent DMA transfers between the peripherals and on-chip dual-port RAM.

The combination of large on-chip RAM, low power dissipation, fast instruction cycle times, and efficient I/O management makes the DSP16210 an ideal solution in a variety of emerging applications.

1. *Motorola* is a registered trademark of Motorola, Inc.
2. *Intel* is a registered trademark of Intel Corporation.
3. *IEEE* is a registered trademark of The Institute of Electrical and Electronics Engineers, Inc.

Table of Contents

Contents	Page		Contents	Page

Table of Contents (continued)

List of Figures

Figures   **Page**

List of Figures (continued)

Figure **Page**

List of Tables

Tables   **Page**

List of Tables (continued)

Tables **Page**

List of Tables (continued)

Tables **Page**

Notation Conventions

The following notation conventions apply to this data sheet:

lower-case	Registers that are directly writable or readable by DSP16210 core instructions are lower-case.
UPPER-CASE	Device flags, I/O pins, and registers that are not directly writable or readable by DSP16210 core instructions are upper-case.
boldface	Register names and DSP16210 core instructions are printed in boldface when used in text descriptions.
italics	Documentation variables that are replaced are printed in italics.
courier	DSP16210 program examples are printed in courier font.
[ ]	Square brackets enclose a range of numbers that represents multiple bits in a single register or bus. The range of numbers is delimited by a colon. For example, **ioc**[7:5] are bits 7—5 of the program-accessible **ioc** register.
⟨ ⟩	Angle brackets enclose a list of items delimited by commas or a range of items delimited by a dash (—), one of which is selected if used in an instruction. For example, **ICSB⟨0—7⟩** represents the eight memory-mapped registers **ICSB0**, **ICSB1**, . . . , **ICSB7**, and the general instruction **aTE⟨h,l⟩ = RB** can be replaced with **a0h = timer0**.

Hardware Architecture

The DSP16210 device is a 16-bit fixed-point programmable digital signal processor (DSP). The DSP16210 consists of a DSP16000 core together with on-chip memory and peripherals. Advanced architectural features with an expanded instruction set deliver a dramatic increase in performance for signal coding algorithms. This increase in performance together with an efficient design implementation results in an extremely cost- and power-efficient solution for wireless and multimedia applications.

DSP16210 Architectural Overview

Figure 1 on page 10 shows a block diagram of the DSP16210. The following blocks make up this device.

DSP16000 Core

The DSP16000 core is the signal-processing engine of the DSP16210. It is a modified Harvard architecture with separate sets of buses for the instruction/coefficient (X-memory) and data (Y-memory) spaces. Each set of buses has 20 bits of address and 32 bits of data. The core contains data and address arithmetic units and control for on-chip memory and peripherals.

Clock Synthesizer (PLL)

The DSP16210 exits device reset with an input clock (CKI) as the source for the internal clock (CLK). An on-chip clock synthesizer (PLL) that runs at a frequency multiple of CKI can also be used to generate CLK. The clock synthesizer is deselected and powered down on reset. For low-power operation, an internally generated slow clock can drive the DSP.

The clock synthesizer and other programmable clock sources are discussed in Clock Synthesis beginning on page 56. The use of these programmable clock sources for power management is discussed in Power Management beginning on page 61.

Dual-Port RAM (DPRAM)

This block contains 60 banks (banks 1—60) of zero wait-state memory. Each bank consists of 1K 16-bit words and has separate address and data ports to the instruction/coefficient (X-memory) and data (Y-memory) spaces. DPRAM is organized into even and odd interleaved banks where each even/odd pair is a 32-bit wide module (see Figure 4 on page 25 for details). Placing instructions and Y-memory data in the same 2K module of DPRAM is not supported and may cause undefined results.

A program can be downloaded from slow off-chip memory into DPRAM, and then executed without wait-states. DPRAM is also useful for improving convolution performance in cases where the coefficients are adaptive. Since DPRAM can be downloaded through the JTAG port, full-speed remote in-circuit emulation is possible.

Hardware Architecture (continued)
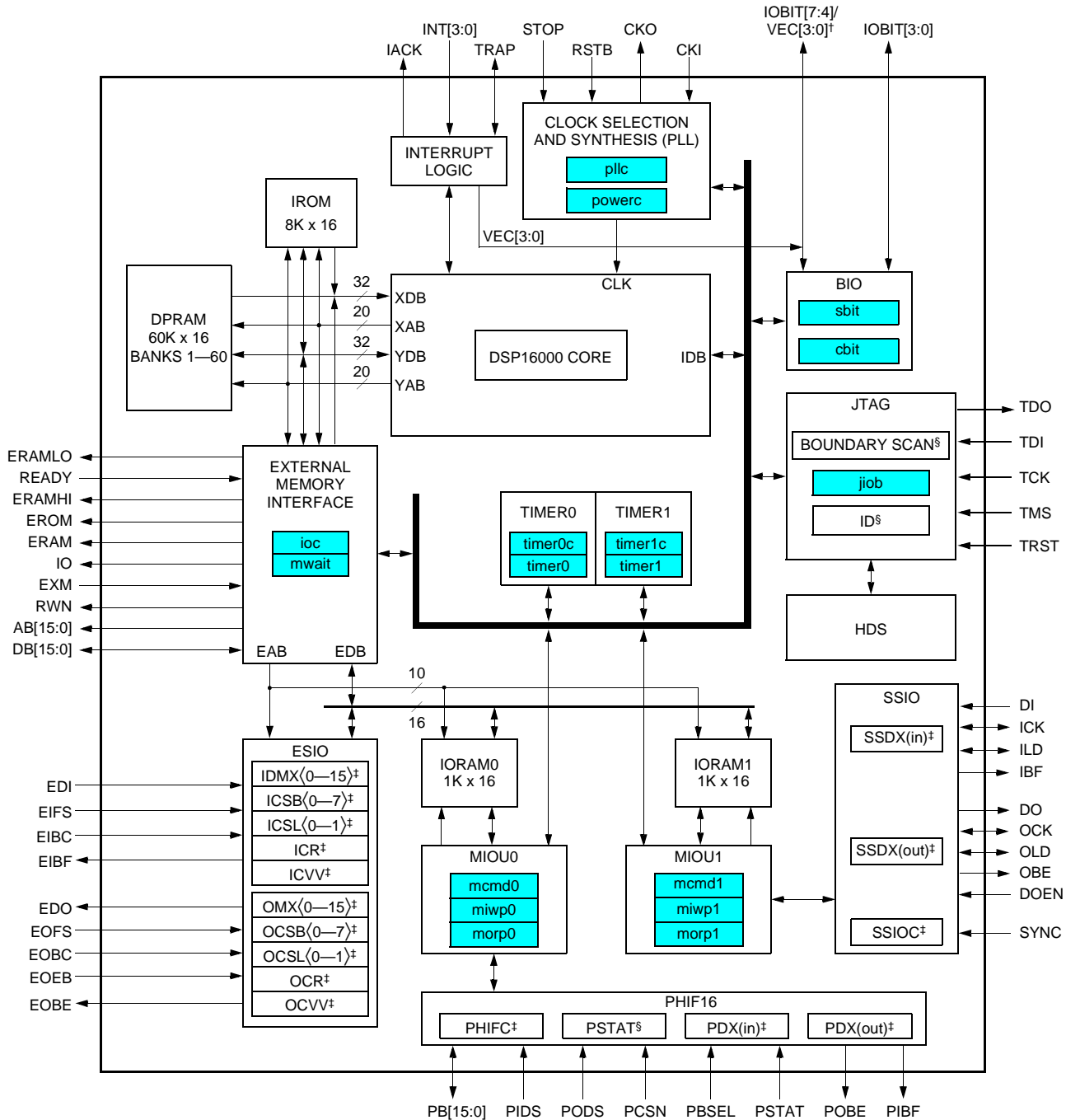
DSP16210 Architectural Overview (continued)

Figure 1. DSP16210 Block Diagram

† VEC0 corresponds to IOBIT7, VEC1 corresponds to IOBIT6, VEC2 corresponds to IOBIT5, and VEC3 corresponds to IOBIT4.
‡ These registers are not directly program accessible.
§ These registers are accessible through pins only.

Hardware Architecture (continued)

DSP16210 Architectural Overview (continued)

Table 1. DSP16210 Block Diagram Legend

Symbol	Description
BIO	Bit I/O Unit
cbit	BIO Control Register
CLK	Internal Clock Signal
DPRAM	Dual-port Random-Access Memory
EAB	EMI Address Bus
EDB	EMI Data Bus
ESIO	Enhanced Serial I/O Unit
HDS	Hardware Development System Unit
ICR	ESIO Input Control Register
ICSB⟨0—7⟩	ESIO Input Channel Start Bit Registers
ICSL⟨0—1⟩	ESIO Input Channel Sample Length Registers
ICVV	ESIO Input Channel Valid Vector Register
IDMX⟨0—15⟩	ESIO Input Demultiplexer Registers
ID	JTAG Device Identification Register
IDB	Internal Data Bus
ioc	I/O Configuration Register
IORAM0	Internal I/O RAM 0: Shared with MIOU0
IORAM1	Internal I/O RAM 1: Shared with MIOU1
IROM	Internal Boot Read-Only Memory
jiob	JTAG Test Register
JTAG	JTAG Test Port
MIOU0	Modular I/O Unit 0: Controls PHIF16
mcmd0	MIOU0 Command Register
miwp0	MIOU0 IORAM0 Input Write Pointer
morp0	MIOU0 IORAM0 Output Read Pointer
MIOU1	Modular I/O Unit 1: Controls SSIO
mcmd1	MIOU1 Command Register
miwp1	MIOU1 IORAM1 Input Write Pointer
morp1	MIOU1 IORAM1 Output Read Pointer
mwait	EMI Configuration Register
OCR	ESIO Output Control Register
OCSB⟨0—7⟩	ESIO Output Channel Start Bit Registers
OCSL⟨0—1⟩	ESIO Output Channel Sample Length Registers
OCVV	ESIO Output Channel Valid Vector Register
OMX⟨0—15⟩	ESIO Output Multiplexer Registers
PDX(in)	PHIF16 Input Register; Readable by MIOU0
PDX(out)	PHIF16 Output Register; Writable by MIOU0
PHIF16	16-bit Parallel Host Interface
PHIFC	PHIF16 Control Register: Programmed Through MIOU0
PLL	Phase-Lock Loop
pllc	Phase-Lock Loop Control Register

Hardware Architecture (continued)

DSP16210 Architectural Overview (continued)

Table 1. DSP16210 Block Diagram Legend (continued)

Symbol	Description
powerc	Power Control Register
PSTAT	PHIF16 Status Register
sbit	BIO Status/Control Register
SSDX(in)	SSIO Input Register; Readable by MIOU1
SSDX(out)	SSIO Output Register; Writable by MIOU1
SSIO	Simple Serial I/O Unit
SSIOC	SSIO Control Register: Programmed Through MIOU1
TIMER0	Programmable Timer 0
timer0	Timer Running Count Register for TIMER0
timer0c	Timer Control Register for TIMER0
TIMER1	Programmable Timer 1
timer1	Timer Running Count Register for TIMER1
timer1c	Timer Control Register for TIMER1
XAB	X-Memory Space Address Bus
XDB	X-Memory Space Data Bus
YAB	Y-Memory Space Address Bus
YDB	Y-Memory Space Data Bus

Internal Boot ROM (IROM)

The DSP16210 includes a boot ROM that contains hardware development code and boot routines. The boot routines are available for use by the programmer and are detailed in DSP16210 Boot Routines beginning on page 126.

IORAM and Modular I/O Units (MIOUs)

IORAM storage consists of two 1 Kword banks of memory, IORAM0 and IORAM1. Each IORAM bank has two 16-bit data and two 10-bit address ports; an IORAM bank can be shared with the core and a modular I/O unit (MIOU) to implement a DMA-based I/O system. IORAM supports concurrent core execution and MIOU I/O processing.

MIOU0 (controls PHIF16) is attached to IORAM0; MIOU1 (controls SSIO) is attached to IORAM1. Portions of IORAM not dedicated to I/O processing can be used as general-purpose data storage.

Placing instructions and Y-memory data in the same IORAM is not supported and may cause undefined results.

The IORAMs and MIOUs are described in detail in Modular I/O Units (MIOUs) beginning on page 42.

External Memory Interface (EMI)

The EMI connects the DSP16210 to external memory and I/O devices. It multiplexes the two sets of core buses (X and Y) onto a single set of external buses—a 16-bit address bus (AB[15:0]) and 16-bit data bus (DB[15:0]). These external buses can access external RAM (ERAMHI/ERAMLO), external ROM (EROM), and memory-mapped I/O space (IO).

The EMI also manages the on-chip IORAM and ESIO storage. It multiplexes the two sets of core buses onto a single set of internal buses—a 10-bit address bus (EAB[9:0]) and 16-bit data bus (EDB[15:0])—to interface to the IORAMs and ESIO memory-mapped registers.

Instructions can transparently reference external memory, IORAM, and ESIO storage from either set of core buses. The EMI automatically translates a single 32-bit access into two 16-bit accesses and vice versa.

The EMI is described in detail in External Memory Interface (EMI) beginning on page 27.

Hardware Architecture (continued)

DSP16210 Architectural Overview (continued)

Bit I/O (BIO) Unit

The BIO unit provides convenient and efficient monitoring and control of eight individually configurable pins (IOBIT[7:0]). When configured as outputs, the pins can be individually set, cleared, or toggled. When configured as inputs, individual pins or combinations of pins can be tested for patterns. Flags returned by the BIO are testable by conditional instructions. See Bit Input/Output Unit (BIO) beginning on page 52 for more details.

Enhanced Serial I/O (ESIO) Unit

The ESIO is a programmable, hardware-managed, passive, double-buffered full-duplex serial input/output port designed to support glueless multichannel I/O processing on a TDM (time-division multiplex) highway. In simple mode, the ESIO supports data rates of up to 26 Mbits/s for a single channel with either 8-bit or 16-bit data lengths. In frame mode, the ESIO processes up to 16 logical TDM channels with a data rate of up to 8.192 Mbits/s. For more information on the ESIO, see Enhanced Serial I/O (ESIO) Unit beginning on page 32.

Simple Serial I/O (SSIO) Unit

The SSIO unit offers a full-duplex, double-buffered external channel that operates at up to 26 Mbits/s. Commercially available codecs and time-division multiplex channels can be interfaced to the SSIO with few, if any, additional components.

The SSIO is a DMA peripheral managed by MIOU1. See Simple Serial I/O (SSIO) Unit beginning on page 49 for more information.

Parallel Host Interface (PHIF16)

The PHIF16 is a DMA peripheral managed by MIOU0. It is a passive 16-bit parallel port that can be configured to interface to either an 8- or 16-bit external bus containing other Lucent Technologies DSPs, microprocessors, or off-chip I/O devices. The PHIF16 port supports either *Motorola* or *Intel* protocols.

When operating in the 16-bit external bus configuration, PHIF16 can be programmed to swap high and low bytes. When operating in 8-bit external bus configuration, PHIF16 is accessed in either an 8-bit or 16-bit logical mode. In 16-bit mode, the host selects either a high or low byte access; in 8-bit mode, only the low byte is accessed.

Additional software-programmable features allow for a glueless host interface to microprocessors (see Parallel Host Interface (PHIF16) beginning on page 49).

Timers

The two timers can be used to provide an interrupt, either single or repetitive, at the expiration of a programmed interval. More than nine orders of magnitude of interval selection are provided. The timers can be stopped and restarted at any time under program control. See Timers beginning on page 53 for more information.

Test Access Port (JTAG)

The DSP16210 provides a test access port that conforms to *IEEE* 1149.1 (JTAG). The JTAG port provides boundary scan test access and also controls the Hardware Development System (HDS). See JTAG Test Port beginning on page 54 for details.

Hardware Development System (HDS)

The HDS is an on-chip hardware module available for debugging assembly-language programs that execute on the DSP16000 core in real-time. The main capability of the HDS is in allowing controlled visibility into the core's state during program execution. The HDS is enhanced with powerful debugging capabilities such as complex breakpointing conditions, multiple data/address watchpoint registers, and an intelligent trace mechanism for recording discontinuities. See Hardware Development System (HDS) beginning on page 54 for details.

Pin Multiplexing

The upper four BIO pins (IOBIT[7:4]) are multiplexed with the vectored interrupt identification pins (VEC[3:0]). Specifically, VEC0 is multiplexed with IOBIT7, VEC1 with IOBIT6, VEC2 with IOBIT5, and VEC3 with IOBIT4. VEC[3:0] are connected to the package pins and IOBIT[7:4] are disconnected immediately after device reset. To select IOBIT[7:4] to be connected to these pins, the program must set EBIO (bit 8 of the **ioc** register).

Hardware Architecture (continued)

DSP16000 Core Architectural Overview

See the *DSP16000 Digital Signal Processor Core* Information Manual for a complete description of the DSP16000 core. Figure 2 on page 16 shows a block diagram of the core that consists of four major blocks: System Control and Cache (SYS), Data Arithmetic Unit (DAU), Y-Memory Space Address Arithmetic Unit (YAAU), and X-Memory Space Address Arithmetic Unit (XAAU). Bits within the **auc0** and **auc1** registers configure the DAU mode-controlled operations.

System Control and Cache (SYS)

This section consists of the control block and the cache.

The control block provides overall system coordination that is mostly invisible to the user. The control block includes an instruction decoder and sequencer, a pseudorandom sequence generator (PSG), an interrupt and trap handler, a wait-state generator, and low-power standby mode control logic. An interrupt and trap handler provides a user-locatable vector table and three levels of user-assigned interrupt priority.

SYS contains the **alf** register, which is a 16-bit register that contains AWAIT, a power-saving standby mode bit, and peripheral flags. The **inc0** and **inc1** registers are 20-bit interrupt control registers, and **ins** is a 20-bit interrupt status register.

Programs use the instruction cache to store and execute repetitive operations such as those found in an FIR or IIR filter section. The cache can contain up to 31 16-bit and 32-bit instructions. The code in the cache can repeat up to $2^{16} - 1$ times without looping overhead. Operations in the cache that require a coefficient access execute at twice the normal rate because the XAAU and its associated bus are not needed for fetching instructions. The cache greatly reduces the need for writing in-line repetitive code and, therefore, reduces instruction/coefficient memory size requirements. In addition, the use of cache reduces power consumption because it eliminates memory accesses for instruction fetches.

The **cloop** register controls the cache loop count. The **cstate** register contains the current state of the cache. The 32-bit **csave** register holds the opcode of the instruction following the loop instruction in X-memory. The cache provides a convenient, low-overhead looping structure that is interruptible, savable, and restorable. The cache is addressable in both the X and Y memory spaces. An interrupt or trap handling routine can save and restore **cloop**, **cstate**, **csave**, and the contents of the cache.

Data Arithmetic Unit (DAU)

The DAU is a power-efficient, dual-MAC (multiply/accumulate) parallel-pipelined structure that is tailored to communications applications. It can perform two double-word (32-bit) fetches, two multiplications, and two accumulations in a single instruction cycle. The dual-MAC parallel pipeline begins with two 32-bit registers, **x** and **y**. The pipeline treats the 32-bit registers as four 16-bit signed registers if used as input to two signed 16-bit x 16-bit multipliers. Each multiplier produces a full 32-bit result stored into registers **p0** and **p1**. The DAU can direct the output of each multiplier to a 40-bit ALU or a 40-bit 3-input ADDER. The ALU and ADDER results are each stored in one of eight 40-bit accumulators, **a0** through **a7**. The ALU includes an ACS (add/compare/select) function for Viterbi decoding. The DAU can direct the output of each accumulator to the ALU/ACS, the ADDER, or a 40-bit BMU (bit manipulation unit).

The ALU implements addition, subtraction, and various logical operations. To support Viterbi decoding, the ALU has a split mode in which it computes two simultaneous 16-bit additions or subtractions. This mode, available in a specialized dual-MAC instruction, is used to compute the distance between a received symbol and its estimate.

The ACS provides the add/compare/select function required for Viterbi decoding. This unit provides flags to the traceback encoder for implementing mode-controlled side-effects for ACS operations. The source operands for the ACS are any two accumulators, and results are written back to one of the source accumulators.

The BMU implements barrel-shift, bit-field insertion, bit-field extraction, exponent extraction, normalization, and accumulator shuffling operations. **ar0** through **ar3** are auxiliary registers whose main function is to control BMU operations.

The user can enable overflow saturation to affect the multiplier output and the results of the three arithmetic units. Overflow saturation can also affect an accumulator value as it is transferred to memory or other register. These features accommodate various speech coding standards such as GSM-FR, GSM-HR, and GSM-EFR. Shifting in the arithmetic pipeline occurs at several stages to accommodate various standards for mixed- and double-precision multiplications.

Hardware Architecture (continued)

DSP16000 Core Architectural Overview
(continued)

The DAU contains control and status registers **auc0**, **auc1**, **psw0**, **psw1**, **vsw**, and **c0—c2**.

The arithmetic unit control registers **auc0** and **auc1** select or deselect various modes of DAU operation. These modes include scaling of the products, saturation on overflow, feedback to the **x** and **y** registers from accumulators **a6** and **a7**, simultaneous loading of **x** and **y** registers with the same value (used for single-cycle squaring), and clearing the low half of registers when loading the high half to facilitate fixed-point operations.

The processor status word registers **psw0** and **psw1** contain flags set by ALU/ACS, ADDER, or BMU operations. They also include information on the current status of the interrupt controller.

The **vsw** register is the Viterbi support word associated with the traceback encoder. The traceback encoder is a specialized block for accelerating Viterbi decoding. It performs mode-controlled side-effects for three MAC instruction group compare functions: **cmp0( )**, **cmp1( )**, and **cmp2( )**. The **vsw** register controls the modes. The side-effects allow the DAU to store, with no overhead, state information necessary for traceback decoding. Side-effects use the **c1** counter, the **ar0** and **ar1** auxiliary registers, and bits 1 and 0 of **vsw**.

The **c1** and **c0** counters are 16-bit signed registers used to count events such as the number of times the program has executed a sequence of code. The **c2** register is a holding register for counter **c1**. Conditional instructions control these counters and provide a convenient method of program looping.

Y-Memory Space Address Arithmetic Unit (YAAU)

The YAAU supports high-speed, register-indirect, data memory addressing and postincrementing of the address register. Eight 20-bit pointer registers (**r0—r7**) store read or write addresses for the Y-memory space. Two sets of 20-bit registers (**rb0** and **re0**; **rb1** and **re1**) define the upper and lower boundaries of two zero-overhead circular buffers for efficient filter implementations. The **j** and **k** registers are two 20-bit signed registers that are used to hold user-defined postincrement values for **r0—r7**. Fixed increments of +1, −1, 0, +2, and −2 are also available. (Postincrement options 0 and −2 are not available for some specialized transfers. See the *DSP16000 Digital Signal Processor Core* Information Manual for details.)

The YAAU includes a 20-bit stack pointer (**sp**). The data move group includes a set of stack instructions that consists of push, pop, stack-relative, and pipelined stack-relative operations. The addressing mode used for the stack-relative instructions is register-plus-displacement indirect addressing (the displacement is optional). The displacement is specified as either an immediate value as part of the instruction or a value stored in **j** or **k**. The YAAU computes the address by adding the displacement to **sp** and leaves the contents of **sp** unchanged. The data move group also includes instructions with register-plus-displacement indirect addressing for the pointer registers **r0—r6** in addition to **sp**.

The data move group of instructions includes instructions for loading and storing any YAAU register from or to memory or another core register. It also includes instructions for loading any YAAU register with an immediate value stored with the instruction. The pointer arithmetic group of instructions allows adding of an immediate value or the contents of the **j** or **k** register to any YAAU pointer register and storing the result to any YAAU register.

X-Memory Space Address Arithmetic Unit (XAAU)

The XAAU contains registers and an adder that control the sequencing of instructions in the processor. The program counter (**PC**) automatically increments through the instruction space. The interrupt return register **pi**, the subroutine return register **pr**, and the trap return register **ptrap** are automatically loaded with return addresses that direct the return to main program execution from interrupt service routines, subroutines, and trap service routines, respectively. High-speed, register-indirect, read-only memory addressing with postincrementing is done with the **pt0** and **pt1** registers. The signed registers **h** and **i** are used to hold a user-defined signed postincrement value. Fixed postincrement values of 0, +1, −1, +2, and −2 are also available. (Postincrement options 0 and −2 are available only if the target of the data transfer is an accumulator vector. *See the DSP16000 Digital Signal Processor Core* Information Manual for details.)

The data move group of instructions includes instructions for loading and storing any XAAU register from or to memory or another core register. It also includes instructions for loading any XAAU register with an immediate value stored with the instruction.

vbase is the 20-bit vector base offset register. The user programs this register with the base address of the interrupt and trap vector table.

Hardware Architecture (continued)

DSP16000 Core Architectural Overview (continued)
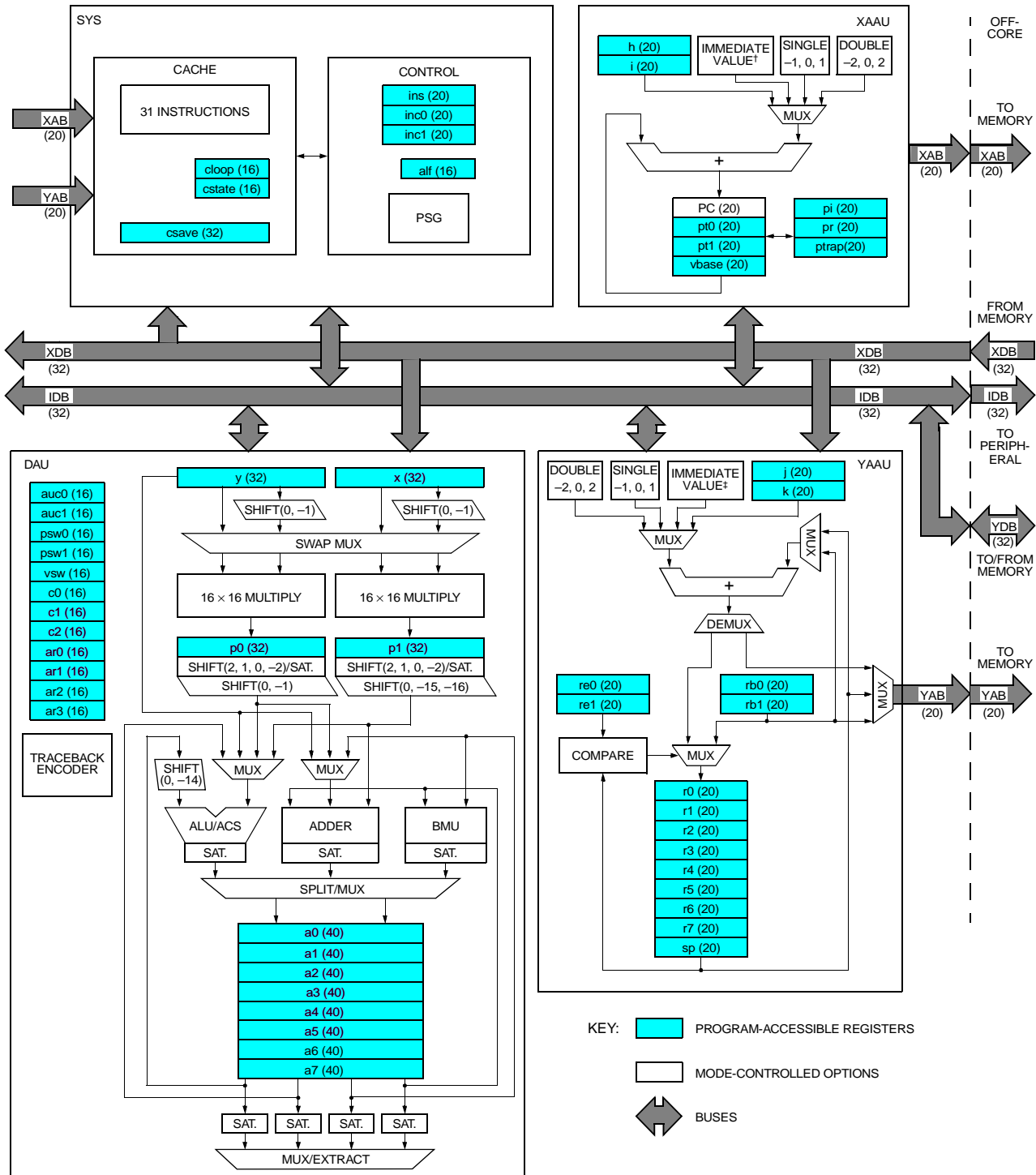
† Associated with **PC**-relative branch addressing.

‡ Associated with register-plus-displacement indirect addressing.

Figure 2. DSP16000 Core Block Diagram

Hardware Architecture (continued)

DSP16000 Core Architectural Overview (continued)

Table 2. DSP16000 Core Block Diagram Legend

Symbol	Name
16 x 16 MULTIPLY	16-bit x 16-bit Multiplier
a0—**a7**	Accumulators 0—7
ADDER	3-input 40-bit Adder/Subtractor
alf	AWAIT and Flags
ALU/ACS	40-bit Arithmetic Logic Unit and Add/Compare/Select Function—used in Viterbi decoding
ar0—**ar3**	Auxiliary Registers 0—3
auc0, **auc1**	Arithmetic Unit Control Registers
BMU	40-bit Bit Manipulation Unit
c0, **c1**	Counters 0 and 1
c2	Counter Holding Register
cloop	Cache Loop Count
COMPARE	Comparator
csave	Cache Save Register
cstate	Cache State Register
DAU	Data Arithmetic Unit
h	Pointer Postincrement Register for the X-Memory Space
i	Pointer Postincrement Register for the X-Memory Space
IDB	Internal Data Bus
inc0, **inc1**	Interrupt Control Registers 0 and 1
ins	Interrupt Status Register
j	Pointer Postincrement/Offset Register for the Y-Memory Space
k	Pointer Postincrement/Offset Register for the Y-Memory Space
MUX	Multiplexer
p0, **p1**	Product Registers 0 and 1
PC	Program Counter
pi	Program Interrupt Return Register
pr	Program Return Register
PSG	Pseudorandom Sequence Generator
psw0, **psw1**	Processor Status Word Registers 0 and 1
pt0, **pt1**	Pointers 0 and 1 to X-Memory Space
ptrap	Program Trap Return Register
r0—**r7**	Pointers 0—7 to Y-Memory Space
rb0, **rb1**	Circular Buffer Pointers 0 and 1 (begin address)
re0, **re1**	Circular Buffer Pointers 0 and 1 (end address)
SAT	Saturation
SHIFT	Shifting Operation
sp	Stack Pointer
SPLIT/MUX	Split/Multiplexer—routes the appropriate ALU/ACS, BMU, and ADDER outputs to the appropriate accumulator
SWAP MUX	Swap Multiplexer—routes the appropriate data to the appropriate multiplier input
SYS	System Control and Cache

Hardware Architecture (continued)

DSP16000 Core Architectural Overview (continued)

Table 2. DSP16000 Core Block Diagram Legend (continued)

Symbol	Name
vbase	Vector Base Offset Register
vsw	Viterbi Support Word—associated with the traceback encoder
x	Multiplier Input Register
XAAU	X-Memory Space Address Arithmetic Unit
XAB	X-Memory Space Address Bus
XDB	X-Memory Space Data Bus
y	Multiplier Input Register
YAAU	Y-Memory Space Address Arithmetic Unit
YAB	Y-Memory Space Address Bus
YDB	Y-Memory Space Data Bus

Reset

The DSP16210 has two negative-assertion external reset input pins: RSTB and TRST. RSTB is used to reset the DSP16210. The primary function of TRST is to reset the JTAG controller.

Reset After Powerup or Power Interruption

At initial powerup or if power is interrupted, a reset is required and both TRST and RSTB must be asserted (low) simultaneously for at least seven CKI cycles (see Reset Circuit on page 142 for details). The TRST pin must be asserted even if the JTAG controller is not used by the application. Failure to properly reset the device on powerup or after a power interruption can lead to a loss of communication with the DSP16210 pins.

RSTB Pin Reset

Reset initializes the state of user registers, synchronizes the internal clocks, and initiates code execution. The device is properly reset by asserting RSTB (low) for at least seven CKI cycles. After RSTB is deasserted, there is a delay of several CKI cycles before the device begins executing instructions (see Reset Synchronization on page 143 for details). The DSP16210 samples the state of the EXM pin when RSTB is deasserted to determine whether it boots from IROM at location 0x20000 (EXM = 0) or from EROM at location 0x80000 (EXM = 1). See Reset States on page 113 for the values of the user registers after reset.

Table 3 on page 19 defines the states of the output and bidirectional pins both during and after reset. It does not include the TDO output pin, because its state is not affected by RSTB but by the JTAG controller.

Reset (continued)

RSTB Pin Reset (continued)

Table 3. State of Device Output and Bidirectional Pins During and After Reset

Type	Pin	State of Pin During Reset (RSTB = 0)		State of Pin After Reset (RSTB 0→1)
Output	AB[15:0], EIBF, PIBF, IBF, IACK	3-state		logic low
	EOBE, POBE, OBE	3-state		logic high
	DO	3-state		3-state
	EDO	3-state		3-state
	RWN, EROM, ERAMHI, ERAMLO, ERAM, IO	INT0 = 0 (deasserted)	logic high	logic high
		INT0 = 1 (asserted)	3-state	
	CKO	INT0 = 0 (deasserted)	internal clock (CLK = CKI)†	internal clock (CLK = CKI)†
		INT0 = 1 (asserted)	3-state	
Bidirectional (Input/Output)	VEC[3:0]/IOBIT[7:4]	3-state		logic high‡
	IOBIT[3:0], TRAP, OLD, OCK, ILD, ICK	3-state		configured as input
	DB[15:0], PB[15:0]	3-state		3-state

† During and after reset, the internal clock is selected as the CKI input pin and the CKO output pin is selected as the internal clock.
‡ The **ioc** register (Table 54 on page 99) is cleared after reset, including its EBIO field that controls the multiplexing of the VEC0/IOBIT7, VEC1/IOBIT6, VEC2/IOBIT5, and VEC3/IOBIT4 pins. Therefore, after reset, these pins are configured as the VEC[3:0] outputs, which are initialized as logic high during reset.

JTAG Controller Reset

The recommended method of resetting the JTAG controller is to assert RSTB and TRST simultaneously. An alternative method is to clock TCK through at least five cycles with TMS held high. Both methods ensure that the user has control of the device pins. JTAG controller reset does not initialize user registers, synchronize internal clocks, or initiate code execution unless RSTB is also asserted.

Reset of the JTAG controller places it in the test logic reset (TLR) state. While in the TLR state, the DSP16210 3-states all bidirectional pins, clears all boundary-scan cells for unidirectional outputs, and deasserts (high) all external memory interface enable signals (EROM, ERAM, ERAMHI, ERAMLO, and IO). This prevents logic contention.

Hardware Architecture (continued)

Interrupts and Trap

The DSP16210 supports the following interrupts and traps:

- 15 hardware interrupts with three levels of user-assigned priority.

- 64 software interrupts (**icall IM6** instruction).

- The TRAP input pin. (The TRAP pin is configured as an output only under JTAG control to support HDS multiple-processor debugging.) By default, after reset, the TRAP pin is configured as an input and is connected directly to the core via the PTRAP signal. If the TRAP pin is asserted, the core vectors to a user-supplied trap service routine at location **vbase** + 0x4.

Five pins of the DSP16210 are devoted to signaling interrupt service status. The IACK pin goes high when the core begins to service an interrupt or trap, and goes low three internal clock (CLK) cycles later. Four pins, VEC[3:0], carry a code indicating which of the interrupts or trap is being serviced. Table 4 on page 21 contains the encodings used by each interrupt.

If an interrupt or trap condition arises, a sequence of actions service the interrupt or trap before the DSP16210 resumes regular program execution. The interrupt and trap vectors are in contiguous locations in memory, and the base (starting) address of the 352-word vector table is configurable in the **vbase** register. Table 4 on page 21 describes the vector table. Assigning each interrupt and trap source to a unique location differentiates selection of their service routines. When an interrupt or trap is taken, the core saves the contents of **PC** and vectors execution to the appropriate interrupt service routine (ISR) or trap service routine (TSR).

There are 15 hardware interrupts with three levels of user-assigned priority. Interrupts are globally enabled by executing the **ei** (enable interrupts) instruction and globally disabled by executing the **di** (disable interrupts) instruction. The user assigns priorities and individually disables (masks) interrupts by configuring the **inc0** and **inc1** registers. The **ins** register contains status information for each interrupt. The **psw1** register includes control and status bits associated with the interrupt handler. When an interrupt is taken, the **pi** register holds the interrupt return address.

Software interrupts allow the testing of interrupt routines and their operation when interrupts occur at specific code locations. Programmers and system architects can observe behavior of complex code segments when interrupts occur (e.g., multilevel subroutine nesting, cache loops, etc.).

A trap is similar to an interrupt but has the highest possible priority. Traps cannot be disabled by executing a **di** instruction. Traps do not nest, i.e., a TSR cannot be trapped. The state of the **psw1** register is unaffected by traps. When a trap is taken, the **ptrap** register holds the trap return address.

An interrupt or trap service routine can be either a four-word entry in the vector table or a larger service routine reached via a **goto** instruction in the vector table, in either case. The service routine must end with a **treturn** instruction for traps or an **ireturn** instruction for interrupts. Executing **ireturn** globally enables interrupts (executing **treturn** does not).

Interrupt Registers

The software interrupt and the traps are always enabled and do not have a corresponding bit in the **ins** register. Other vectored interrupts are enabled in the **inc0** and **inc1** registers (Table 5 on page 22) and monitored in the **ins** register (Table 6 on page 22). One of three priority levels for each hardware interrupt can be configured using two consecutive bits of **inc0** or **inc1**. There are two reasons for assigning priorities to interrupts.

- Nesting interrupts, i.e., an interrupt service routine can be interrupted by an interrupt of higher priority.

- Servicing concurrent interrupts according to their priority.

The **ins** register indicates the pending status of each interrupt. When set to 1, the status bits in the **ins** register indicate that an interrupt is pending. An instruction clears an interrupt by writing a one to the corresponding bit in the **ins** register (e.g., **ins = IM20**). Writing a zero to any bit leaves the bit unchanged. The interrupts corresponding to the least significant bits of **ins** are given higher default priority[1] than the interrupts corresponding to the most significant bits of **ins**. The processor must reach an interruptible state (completion of an interruptible instruction) before action is taken on an enabled interrupt. An interrupt is not serviced if it is not enabled.

---

1. Priority is primarily determined by programming the **inc0** and **inc1** registers (Table 5 on page 22). For interrupts with the same programmed priority, the position of their corresponding bits in **ins** determine their relative priority. For example, the EOFE and EIFE interrupts (**ins**[12:11]) default to a higher priority than EOBE and EIBF (**ins**[15:14]).

Hardware Architecture (continued)

Interrupts and Trap (continued)

Table 4. Interrupt and User Trap Vector Table

Vector Description	Vector Address†		Priority	VEC[3:0] Signals‡
	Hexadecimal	**Decimal**		
Reserved	**vbase** + 0x0	**vbase** + 0	—	—
PTRAP (driven by TRAP pin)	**vbase** + 0x4	**vbase** + 4	6—Highest	0xD
UTRAP (reserved for HDS)	**vbase** + 0x8	**vbase** + 8	5	0xE
Reserved	**vbase** + 0xC	**vbase** + 12	—	—
Reserved	**vbase** + 0x10	**vbase** + 16	—	—
MIBF0	**vbase** + 0x14	**vbase** + 20	0—3§	0x1
MOBE0	**vbase** + 0x18	**vbase** + 24	0—3§	0x2
MIBF1	**vbase** + 0x1C	**vbase** + 28	0—3§	0x3
MOBE1	**vbase** + 0x20	**vbase** + 32	0—3§	0x4
INT0	**vbase** + 0x24	**vbase** + 36	0—3§	0x5
INT1	**vbase** + 0x28	**vbase** + 40	0—3§	0x6
INT2	**vbase** + 0x2C	**vbase** + 44	0—3§	0x7
INT3	**vbase** + 0x30	**vbase** + 48	0—3§	0x8
TIME0	**vbase** + 0x34	**vbase** + 52	0—3§	0x9
TIME1	**vbase** + 0x38	**vbase** + 56	0—3§	0xA
EIFE	**vbase** + 0x3C	**vbase** + 60	0—3§	0xB
EOFE	**vbase** + 0x40	**vbase** + 64	0—3§	0xC
ECOL	**vbase** + 0x44	**vbase** + 68	0—3§	0xD
EIBF	**vbase** + 0x48	**vbase** + 72	0—3§	0xE
EOBE	**vbase** + 0x4C	**vbase** + 76	0—3§	0xF
Reserved	**vbase** + 0x50	**vbase** + 80	—	—
Reserved	**vbase** + 0x54	**vbase** + 84	—	—
Reserved	**vbase** + 0x58	**vbase** + 88	—	—
Reserved	**vbase** + 0x5C	**vbase** + 92	—	—
Software Interrupt 0 (**icall 0**)	**vbase** + 0x60	**vbase** + 96	—	0xC
Software Interrupt 1 (**icall 1**)	**vbase** + 0x64	**vbase** + 100	—	0xC
⋮	⋮	⋮	—	0xC
Software Interrupt 62 (**icall 62**)	**vbase** + 0x158	**vbase** + 344	—	0xC
Software Interrupt 63 (**icall 63**)	**vbase** + 0x15C	**vbase** + 348	—	0xC

† **vbase** contains the base address of the 352-word vector table.

‡ The VEC[3:0] signals are multiplexed with the BIO signals IOBIT[7:4] onto the VEC[3:0]/IOBIT[7:4] pins (VEC0 corresponds to IOBIT7, VEC1 corresponds to IOBIT6, VEC2 corresponds to IOBIT5, and VEC3 corresponds to IOBIT4). VEC[3:0] defaults to 0xF (all ones) if the core is not currently servicing an interrupt or a trap.

§ The programmer specifies the relative priority levels 0—3 for hardware interrupts via **inc0** and **inc1** (see the *DSP16000 Digital Signal Processor Core* Information Manual). Level 0 indicates a disabled interrupt. If the core simultaneously recognizes more than one interrupt with the same assigned priority, it services the interrupt with the lowest vector address first.

Hardware Architecture (continued)

Interrupts and Trap (continued)

Table 5. Interrupt Control 0 and 1 (inc0, inc1) Registers

	19—18	17—16	15—14	13—12	11—10
inc0	TIME0[1:0]	INT3[1:0]	INT2[1:0]	INT1[1:0]	INT0[1:0]
inc1	Reserved—write with zero				EOBE[1:0]
	9—8	**7—6**	**5—4**	**3—2**	**1—0**
inc0	MOBE1[1:0]	MIBF1[1:0]	MOBE0[1:0]	MIBF0[1:0]	Reserved
inc1	EIBF[1:0]	ECOL[1:0]	EOFE[1:0]	EIFE[1:0]	TIME1[1:0]

Field†	Value	Description
TIME0[1:0] INT3[1:0] INT2[1:0] INT1[1:0]	00	Disable the selected interrupt (no priority).
INT0[1:0] MOBE1[1:0] MIBF1[1:0] MOBE0[1:0]	01	Enable the selected interrupt at priority 1 (lowest).
MIBF0[1:0] EOBE[1:0] EIBF[1:0] ECOL[1:0]	10	Enable the selected interrupt at priority 2.
EOFE[1:0] EIFE[1:0] TIME1[1:0]	11	Enable the selected interrupt at priority 3 (highest).

† Reset clears all fields to disable all interrupts.

Table 6. Interrupt Status (ins) Register

	19—16			15	14	13	12	11	10
	Reserved			EOBE	EIBF	ECOL	EOFE	EIFE	TIME1
9	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TIME0	INT3	INT2	INT1	INT0	MOBE1	MIBF1	MOBE0	MIBF0	Reserved

Bit	Field†	Value	Description
19—16	Reserved	—	Reserved—write with zero.
15—0	EOBE EIBF ECOL EOFE EIFE TIME1 TIME0	0	Read—corresponding interrupt not pending. Write—no effect.
	INT3 INT2 INT1 INT0 MOBE1 MIBF1 MOBE0 MIBF0	1	Read—corresponding interrupt is pending. Write—clears bit and changes corresponding interrupt status to not pending‡.

† The core clears an interrupt's **ins** bit if it services that interrupt. For interrupt polling, an instruction can explicitly clear an interrupt's **ins** bit by writing a 1 to that bit and a 0 to all other **ins** bits. Writing a 0 to any **ins** bit leaves the bit unchanged.

‡ To clear an interrupt's status, an application writes a 1 to the corresponding bit.

Hardware Architecture (continued)

Interrupts and Trap (continued)

Clearing Interrupts

Writing a 1 to a bit in the **ins** register causes the corresponding interrupt status bit to be cleared to a logic 0. This bit is also automatically cleared by the core when the interrupt is taken, leaving set any other vectored interrupts that are pending. The MIOU and ESIO interrupt requests can be cleared by particular instructions, but there is a latency between the instruction execution and the actual clearing of the interrupt request (see the section below).

Interrupt Request Clearing Latency

As a consequence of pipeline delay, there is a minimum latency (number of instruction cycles) between the time a peripheral interrupt clear instruction is executed for an MIOU or ESIO interrupt and the corresponding interrupt request is actually cleared. These latencies are described in Table 7, and are significant when implementing ISRs or I/O polling loops. See Modular I/O Units (MIOUs) beginning on page 42 and Enhanced Serial I/O (ESIO) Unit beginning on page 32 for details on these interrupts.

Table 7. Interrupt Request Clearing Latency

Interrupt Clear Instruction†	Subsequent Instruction†	Latency (Cycles)	Example‡	

mcmd⟨0,1⟩ = ⟨ILEN_UP, OLEN_UP, RESET⟩	**ireturn**
	
	(return from interrupt service routine)
		4	`mcmd0=0x4010`
			`4*nop`
			`ireturn`
				ILEN_UP command clears MIBF0 request. Four **nop**s are needed to avoid unintentional re-entry into ISR.

	ins = ⟨REG, MEM⟩
	
	(clear interrupt pending bit within a polling routine)
		6	`mcmd1=0x6000`
			`6*nop`
			`ins=0x00008`
			`a0=ins`
				RESET command clears MIBF1 request and sets MOBE1 request. Six **nop**s are needed before MIBF1 bit in **ins** can be cleared.

REG = MEM (MEM is IDMX⟨0—15⟩)
 	
or
 	
MEM = REG (MEM is **ICR**)
(Bit 4 of REG is one, setting IRESET field.)
	ireturn
	
	(return from interrupt service routine)
		2	`a5h=*r0`
			`2*nop`
			`ireturn`
				r0 is 0xe0000. **a5h = *r0** reads **IDMX0** and clears EIBF request. Two **nop**s are needed to avoid unintentional re-entry into ISR.

	ins = ⟨REG, MEM⟩
	
	(clear interrupt pending bit within a polling routine)
		4	`*r5=a1h`
			`4*nop`
			`ins=0x04800`
			`a3=ins`
				r5 is 0xe001A (***r5** is **ICR**). Bit 4 of **a1h** is one. Four **nop**s are needed before EIBF or EIFE bits in **ins** can be cleared.

MEM = REG (MEM is OMX⟨0—15⟩)
 	
or
 	
MEM = REG (MEM is **OCR**)
(Bit 4 or bit 7 of REG is one, setting ORESET or CRESET field)
	ireturn
	
	(return from interrupt service routine)
		2	`*r1=a4h`
			`2*nop`
			`ireturn`
				r1 is 0xe003A (***r1** is **OCR**). Bit 4 of **a4h** is one, causing the clearing of EOBE, EOFE, and ECOL requests. Two **nop**s are needed to avoid unintentional re-entry into ISR.

	ins = ⟨REG, MEM⟩
	
	(clear interrupt pending bit within a polling routine)
		4	`*r6=a3h`
			`4*nop`
			`ins=0x08000`
			`a3=ins`
				r6 is 0xe0020. ***r6 = a3h** writes **OMX0** and clears EOBE request. Four **nop**s are needed before EOBE bit in **ins** can be cleared.

† Key to these columns: REG is any register. MEM is a memory location. ILEN_UP, OLEN_UP, or RESET is a value (immediate, register contents, or memory location contents) such that bits 15:12 are 0x4, 0x5, or 0x6, respectively.

‡ The **nop** and multiple **nop** instructions in the examples can be replaced by any instruction(s) that takes an equal or greater number of execution cycles than the **nop** instruction(s).

Hardware Architecture (continued)

Interrupts and Trap (continued)

INT[3:0] and TRAP Pins

The DSP16210 provides four interrupt pins INT[3:0]. TRAP is a bidirectional pin. At reset TRAP is configured as an input to the processor. Asserting the TRAP pin forces a pin trap. The trap mechanism is used to rapidly gain control of the processor for asynchronous time-critical event handling (typically for catastrophic error recovery). A separate vector, PTRAP, is provided for the pin trap (see Table 4 on page 21). Traps cannot be disabled.

Referring to the timing diagram in Figure 3, the INT[3:0] or TRAP pin is asserted for a minimum of two cycles. The pin is synchronized and latched on the next falling edge of CLK. A minimum of four cycles later, the interrupt or trap gains control of the core and the core branches to the interrupt service routine (ISR) or trap service routine (TSR). The actual number of cycles until the interrupt or trap gains control of the core depends on the number of wait-states incurred by the interrupted or trapped instruction. The DSP16210 drives a value (see Table 4 on page 21) onto the VEC[3:0] pins and asserts the IACK pin.

Low-Power Standby Mode

The DSP16210 has a power-saving standby mode in which the internal core clock stretches indefinitely until the core receives an interrupt or trap request. A minimum amount of core circuitry remains active in order to process the incoming interrupt. The clocks to the peripherals are unaffected and the peripherals continue to operate during standby mode. The program places the core in standby mode by setting the AWAIT bit (bit 15) of the **alf** register (**alf** = 0x8000). After the AWAIT bit is set, one additional instruction is executed before the standby mode is entered. When an interrupt occurs, core hardware resets AWAIT, and normal core processing is resumed.

The MIOUs remain operational even in standby mode. Their clocks remain running and they continue any DMA activity.

Two **nop** instructions should be programmed after the AWAIT bit is set. The first **nop** (one cycle) is executed before sleeping; the second is executed after the interrupt signal awakens the DSP and before the interrupt service routine is executed.

Power consumption can be further reduced by activating other available low-power modes. See Power Management beginning on page 61 for information on these other modes.
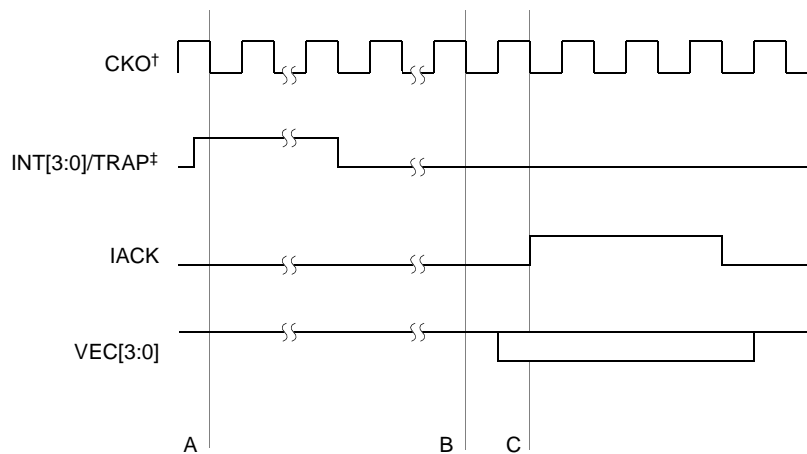
† CKO is programmed to be CLK.
‡ The INT[3:0] or TRAP pin must be held high for a minimum of two cycles.

Notes:
A. The DSP16210 synchronizes and latches the INT[3:0] or TRAP.

B. A minimum four-cycle delay before the core services the interrupt or trap (executes instructions starting at the vector location). For a trap, the core executes a maximum of three instructions before it services the trap.

Figure 3. INT[3:0] and TRAP Timing

Hardware Architecture (continued)

Memory Maps

Figure 5 shows the DSP16210 X-memory space memory map (XMAP). Figure 6 on page 26 shows the DSP16210 Y-memory space memory maps (YMAP0 and YMAP1). Instructions differentiate between the X- and Y-memory spaces by the addressing unit (i.e., the set of pointers) used for the access and not by the physical memory accessed. Although the memories are 16-bit word-addressable, data or instruction widths can be either 16 bits or 32 bits and the internal memories can be accessed 32 bits at a time. The internal DPRAM is organized into even and odd interleaved banks as shown in Figure 4. The core data buses (XDB and YDB) are 32 bits wide, so the core can access 32-bit DPRAM data that has an aligned (even) address in a single cycle.
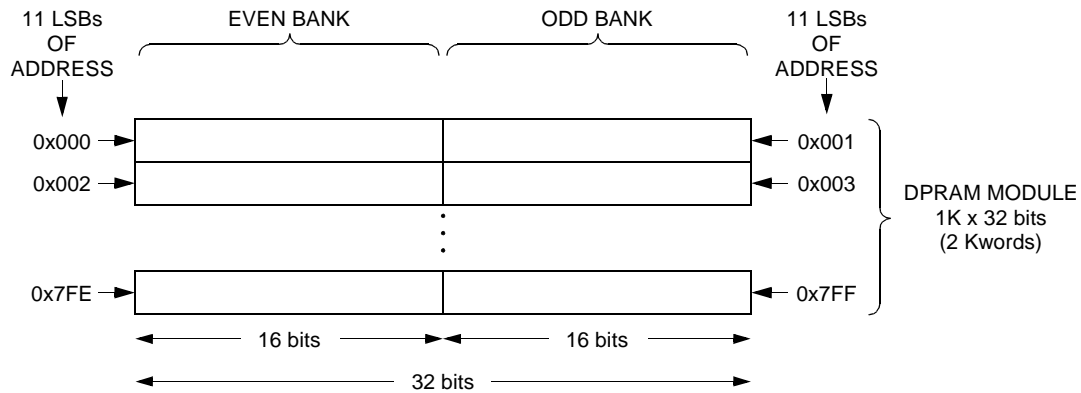
Figure 4. Interleaved Internal DPRAM
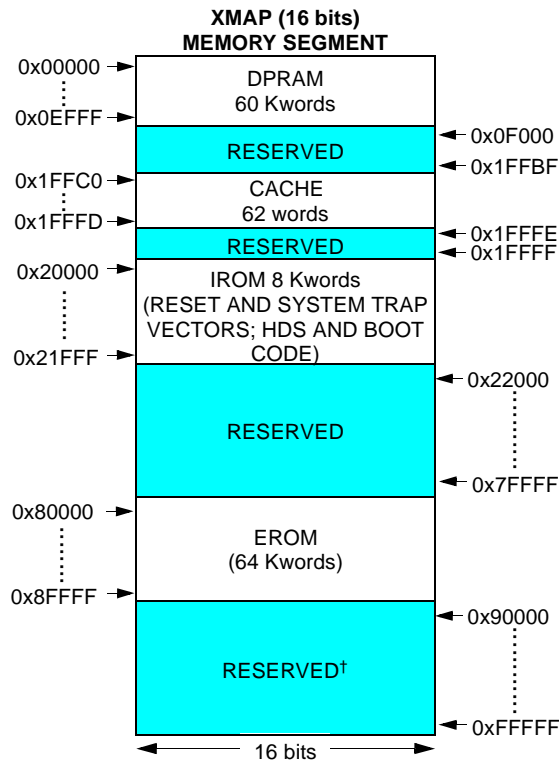
† These locations are modularly mapped into the previous segment (EROM). For example, location 0xA0000 maps to location 0x80000.

Figure 5. X-Memory Space Memory Map

Hardware Architecture (continued)

Memory Maps (continued)

YMAP0 (16 bits)
MEMORY SEGMENT
ioc[WEROM] = 0

Address	Segment
0x00000 – 0x0EFFF	DPRAM 60 Kwords
0x0F000 – 0x1FFBF	RESERVED
0x1FFC0 – 0x1FFFD	CACHE 62 words
0x1FFFE – 0x7FFFF	RESERVED
0x80000 – 0x8FFFF	ERAMLO (64 Kwords)
0x90000 – 0x9FFFF	ERAMHI (64 Kwords)
0xA0000 – 0xAFFFF	IO (64 Kwords)
0xB0000 – 0xBFFFF	RESERVED‡
0xC0000 – 0xC03FF	IORAM0† (1 Kword)
0xC0400 – 0xCFFFF	RESERVED‡ (63 Kwords)
0xD0000 – 0xD03FF	IORAM1† (1 Kword)
0xD0400 – 0xDFFFF	RESERVED‡ (63 Kwords)
0xE0000 – 0xE003F	ESIO† (64 words)
0xE0040 – 0xEFFFF	RESERVED‡ (65,504 words)
0xF0000 – 0xFFFFF	RESERVED‡

16 bits

YMAP1 (16 bits)
MEMORY SEGMENT
ioc[WEROM] = 1

Address	Segment
0x00000 – 0x0EFFF	DPRAM 60 Kwords
0x0F000 – 0x1FFBF	RESERVED
0x1FFC0 – 0x1FFFD	CACHE 62 words
0x1FFFE – 0x7FFFF	RESERVED
0x80000 – 0x8FFFF	EROM (64 Kwords)
0x90000 – 0x9FFFF	RESERVED‡
0xA0000 – 0xAFFFF	IO (64 Kwords)
0xB0000 – 0xBFFFF	RESERVED‡
0xC0000 – 0xC03FF	IORAM0† (1 Kword)
0xC0400 – 0xCFFFF	RESERVED‡ (63 Kwords)
0xD0000 – 0xD03FF	IORAM1† (1 Kword)
0xD0400 – 0xDFFFF	RESERVED‡ (63 Kwords)
0xE0000 – 0xE003F	ESIO† (64 words)
0xE0040 – 0xEFFFF	RESERVED‡ (65,504 words)
0xF0000 – 0xFFFFF	RESERVED‡

16 bits

† IORAM0, IORAM1, and ESIO are internal physical memory spaces that are managed by the EMI and are mapped to external memory addresses.

‡ These locations are modularly mapped into the previous segment. For example, locations 0xD0400—0xD07FF map to locations 0xD0000—0xD03FF and location 0xE0040 maps to location 0xE0000.

Figure 6. Y-Memory Space Memory Maps

The external memory data bus (DB) and the EMI data bus (EDB) are 16 bits wide, and therefore, 32-bit accesses to external memory and IORAM are broken into two 16-bit accesses.

Hardware Architecture (continued)

Memory Maps (continued)

The addresses shown in Figures 5 and 6 correspond to the 20-bit core address buses (XAB for the XMAP and YAB for YMAP0/YMAP1). For external memory accesses, these 20-bit addresses are truncated to 16 bits and the external enable pins (EROM, ERAMHI, ERAMLO, and IO) differentiate the 64K segment being accessed. For IORAM accesses, these 20-bit addresses are truncated to 10 bits.

Boot from External ROM

The EXM pin determines from which memory region (EROM or IROM) the DSP16210 executes code following a device reset. EXM is captured by the rising edge of RSTB. If the captured value of EXM is one, the DSP16210 boots from external ROM (EROM—core address 0x80000). Otherwise, the DSP16210 boots from internal IROM (core address 0x20000). See DSP16210 Boot Routines beginning on page 126 for details on booting from IROM.

Data Memory Map Selection

The DSP16210 data memory map selection is based on the value of the WEROM field (bit 4) in the **ioc** register (Table 54 on page 99). If WEROM is set to 0, the YMAP0 data memory map is selected. If WEROM is set to 1, the YMAP1 data memory map is selected. If WEROM is 1, all ERAMLO accesses are redirected to the EROM segment.

External Memory Interface (EMI)

The external memory interface (EMI) manages off-chip memory and on-chip IORAM memory and ESIO storage, collectively referred to as EMI storage.

The EMI multiplexes the two sets of core buses (XAB/XDB and YAB/YDB) onto a single set of external buses—a 16-bit address bus (AB) and 16-bit data bus (DB). It also multiplexes the two sets of core buses onto a single set of internal EMI buses—a 10-bit address bus (EAB) and a 16-bit data bus (EDB)—for access to the IORAM and ESIO storage. The EMI automatically translates 32-bit XDB/YDB accesses into two 16-bit DB/EDB accesses and vice versa. If an instruction accesses EMI storage from both the X side and Y side, the EMI performs the X access first followed by the Y access and the core incurs a conflict wait-state.

The EMI accesses four external memory segments—ERAMHI, ERAMLO, EROM, and IO.

Two control registers are encoded by the user to define the operation of the EMI. Bits 14—0 in **mwait** (Table 58 on page 101) and bits 10 and 7—0 in **ioc** (Table 54 on page 99) apply to the EMI. These programmable features give the designer flexibility in choosing among various external memories.

Latency for Programming mwait and ioc Registers

There is a two instruction cycle latency between an instruction that updates either **ioc** or **mwait** and availability of the new value in the EMI. It is recommended that two **nop**s (or other instructions that do not access external memory) follow each **ioc** or **mwait** update instruction. See the example below:

```
mwait=0x0222/* Modify mwait          */
2*nop       /* Wait for latency       */
a0=*r0      /* OK to perform EMI read */
```

For write operations the EMI buffers the data (see Functional Timing beginning on page 29), software must verify that all pending external write operations have completed before modifying **ioc** or **mwait**. Software ensures that all memory operations have completed by executing an external memory read operation. After the read operation is completed, it is safe to modify **ioc** or **mwait**. See the code segment below for an example:

```
*r1++=a1    /* EMI write.           */
a0=*r2      /* Dummy EMI read.      */
mwait=0x0222/* Safe to modify mwait. */
2*nop       /* Wait for mwait latency. */
```

Note: For the EMI to function properly, the application program **must** adhere to the latency restrictions presented above.

Hardware Architecture (continued)

External Memory Interface (EMI) (continued)

Programmable Access Time

For each of the four external memory segments, the number of cycles to assert the enable can be selected in
mwait (Table 58 on page 101). Within **mwait**, the IATIM[3:0] field specifies the number of cycles to assert the
enable for the IO segment, the YATIM[3:0] field specifies the number of cycles to assert the enable for the ERAMLO
and ERAMHI segments, and the XATIM[3:0] field specifies the number of cycles to assert the enable for the EROM
segment. On device reset, all access time values are initialized to 15 (**mwait** resets to 0x0FFF).

External memory accesses cause the core to incur wait-states. Table 8 on page 28 defines the duration of an
access and the number of wait-states incurred as a function of the programmed access time (IATIM[3:0],
YATIM[3:0], or XATIM[3:0] abbreviated as A). For example, if YATIM[3:0] = 0xB (decimal 11), then the ERAMLO
and ERAMHI enables are asserted for 11 CLK cycles, any accesses to ERAMLO or ERAMHI require 12 CLK
cycles, and the number of wait-states incurred by the core is 12 for read operations and up to and including 12 for
write operations.

Wait-states for write operations can be transparent to the core if subsequent instructions do not access external
memory.

Table 8. Access Time and Wait-States

	Number of CLK Cycles the Enable Pin Is Asserted	Duration of Access	Wait-States Incurred	
			Read	Write
Quantity	IATIM[3:0], YATIM[3:0], or XATIM[3:0] (abbreviated as ATIM)	ATIM + 1	ATIM + 1	up to and including ATIM + 1
Range	1—15	2—16	2—16	0—16

READY Pin Enables

For each of the four external memory segments, **mwait** (Table 58 on page 101) can be programmed to enable or
disable the READY pin. Setting the RDYEN2 bit enables READY for the IO segment, setting the RDYEN1 bit
enables READY for the ERAMLO and ERAMHI segments, and setting the RDYEN0 bit enables READY for the
EROM segment. On device reset, the RDYEN[2:0] bits are cleared, causing the DSP16210 to ignore the READY
pin by default.

Enable Delays

The leading edge of an enable can be delayed to avoid a situation in which two devices drive the data bus
simultaneously. If the leading edge of an enable is delayed, it is guaranteed to be asserted after the RWN signal is
asserted.

Setting DENB2 of **ioc** (Table 54 on page 99) delays the leading edge of the IO enable by approximately one half-
cycle of CLK. Similarly, setting DENB1 delays the leading edge of the ERAM, ERAMHI, and ERAMLO enables,
and setting DENB0 delays the leading edge of the EROM enable. On device reset, the DENB[2:0] bits are cleared,
causing no delay by default.

Memory Map Selection

The WEROM field (**ioc** bit 4) selects either YMAP0 or YMAP1 (see Figure 6 on page 26). If WEROM is set,
YMAP1 is selected and all ERAMLO accesses are mapped to EROM. This allows the EROM segment, which is
normally read-only, to be written. For example, a program could download code or coefficients into the EROM seg-
ment for later use. If WEROM is set, the DENB1 field (**ioc** bit 1) and the RDYEN1 and YATIM[3:0] fields (**mwait**
bits 13 and 7—4) control Y-side accesses to EROM.

Hardware Architecture (continued)

External Memory Interface (EMI) (continued)

RWN Advance

The RWNADV field (**ioc** bit 3) controls the amount of delay from the beginning of a write access to the lowering of the RWN pin. See External Memory Interface under Timing Characteristics and Requirements for details.

CKO Pin Configuration

The CKOSEL[2:0] field (**ioc** bits 7—5) configures the CKO pin as either the internal free-running clock (CLK), the internal free-running clock held high during low-power standby mode, the output of the CKI input buffer, logic zero, or logic one. See Table 54 on page 99.

Write Data Drive Delay

The write data delay (WDDLY) field (**ioc** bit 10) controls the amount of time that the EMI delays driving write data onto the data bus (DB[15:0]). If WDDLY is cleared, the EMI drives the data bus approximately one half-cycle of CLK after the beginning of the access[1]. If WDDLY is set, the EMI drives the data bus approximately one full cycle of CLK after the beginning of the access[1]. As a result, setting WDDLY provides an additional delay of one half-cycle for slower external memory. This additional delay is particularly useful if the external memory's enable is delayed (the corresponding DENB[2:0] bit is set).

If WDDLY is set, both the turn-on and turn-off delays for the data bus are increased[2]. Because the turn-off delay is increased, it may be necessary to set the corresponding DENB[2:0] bit for any segments that are read immediately after writing.

Functional Timing

The following definitions apply throughout:

Low—an electrical level near ground corresponding to logic zero.

High—an electrical level near VDD corresponding to logic one.

Assertion—the changing of a signal to its active value.

Deassertion—the changing of a signal to its inactive value.

EMI Storage—storage that the EMI manages consisting of external memory, IORAM memory, and ESIO memory-mapped registers.

EMI Instruction—a DSP16210 instruction that accesses (reads or writes) EMI storage.

Non-EMI Instruction—a DSP16210 instruction that does not access EMI storage.

CLK Period—the time from rising edge to rising edge of the CLK clock; the duration of one single instruction cycle. All EMI events occur on the rising edge of CLK. It is assumed that the CKO pin is programmed as CLK and the remainder of this section uses the terms CLK and CKO interchangeably.

---

1. The beginning of the access occurs when the EMI drops RWN.
2. The data bus active interval is constant regardless of WDDLY.

Hardware Architecture (continued)

External Memory Interface (EMI) (continued)

Functional Timing (continued)

All DSP16210 external memory read and write operations consist of two parts:

1. **Active Part**: Lasts for the number of cycles programmed in the **mwait** register (IATIM[3:0], XATIM[3:0], or YATIM[3:0]). Begins on a rising edge of CLK (CKO). Immediately after this rising edge:

    a. The DSP16210 asserts the memory segment enable. If the leading edge of the memory segment enable is delayed (the corresponding DENB[2:0] bit of **ioc** is set), the DSP16210 asserts the memory segment enable one-half of a CLK period later.

    b. The DSP16210 places the address on the address bus AB[15:0].

    c. RWN becomes valid (high for a read, low for a write).

    d. For a read operation, the DSP16210 3-states its data bus DB[15:0] drivers. For a write operation, the DSP16210 delays driving the data bus by an interval determined by the WDDLY field (**ioc** bit 10). If WDDLY = 0, the delay is approximately one half-cycle of CLK after RWN goes low. If WDDLY = 1, the delay is approximately one cycle of CLK after RWN goes low.

2. **Finish Part**: Lasts for one cycle. Begins on a rising edge of CLK (CKO). Immediately after this rising edge:

    a. The DSP16210 deasserts the memory segment enable.

    b. For a read operation, the DSP16210 latches the data from DB[15:0]. For a write operation, the DSP16210 continues to drive data onto the data bus for an interval determined by the WDDLY field (**ioc** bit 10). If WDDLY = 0, the DSP16210 drives the bus for approximately one half-cycle of CLK after the beginning of the finish part. If WDDLY = 1, the DSP16210 drives the bus for approximately one cycle of CLK after the beginning of the finish part.

As a consequence of the finish part of each memory operation, contention problems caused by back-to-back assertion of different enables (one instruction with dual accesses) are avoided. Following the finish part, the DSP16210 continues to drive the address bus with the last valid address until the beginning of the next external read or write operation.

If an instruction **reads** from EMI storage, the number of wait-states incurred by the core during execution of that instruction is R. R is computed as:

$$R = R_X + R_Y$$

where: R_X = Number of wait-states incurred from reading external X-memory[1].
         R_Y = Number of wait-states incurred from reading external Y-memory, IORAM memory, or ESIO register.

If an instruction **writes** to EMI storage and is immediately followed by a second EMI instruction, wait-states are incurred by the core during execution of the second[2] instruction. The number of wait-states is W:

where: W = Number of wait-states incurred from writing external Y-memory, IORAM memory, or ESIO register.

---

1. Including possible instruction fetch.
2. Wait-states are incurred by the following instruction and not by the current instruction because the EMI internally buffers write data. In other words, the core does not wait (as it does in the DSP1620) until the write data has been transferred to EMI storage. Instead, the core continues execution while the EMI waits to transfer the data to EMI storage on the next available memory cycle. A subsequent access to EMI storage causes the core to wait until the prior write operation's data has been transferred to storage.

Hardware Architecture (continued)

External Memory Interface (EMI) (continued)

Functional Timing (continued)

Table 9 describes the computation of wait-states for read and write accesses (R_X, R_Y, and W) for each segment of EMI storage, including the IORAM memories and ESIO memory-mapped registers.

Table 9. Wait-States

Access	Segment		Number of Wait-States[†]
X-memory read	EROM	R_X	$(size \times (XATIM[3:0] + 1) + misaligned)$
Y-memory read	ERAMHI or ERAMLO	R_Y	$(size \times (YATIM[3:0] + 1) + misaligned)$
	IO		$(size \times (IATIM[3:0] + 1) + misaligned)$
	IORAM or ESIO		$(size \times 2 + misaligned)$
Y-memory write	ERAMHI or ERAMLO	$W^{‡}$	$(size \times (YATIM[3:0] + 1) + misaligned)$
	IO		$(size \times (IATIM[3:0] + 1) + misaligned)$
	IORAM or ESIO		$(size + misaligned)$

† *size* is one for a 16-bit access and two for a 32-bit access. *misaligned* is one for a misaligned double-word access and zero for a single-word access or an aligned double-word access.
‡ Write wait-states can be transparent to the core if the EMI write instruction is followed by non-EMI instructions.

Write wait-states can be transparent to the core if the instruction that writes EMI storage is followed by non-EMI instructions. If write wait-states are transparent, then the core continues execution while the EMI completes the write operation. For example, the single write external wait-state in the following code segment is transparent and does not stall the core execution:

```
*r0++=a0h   /* r0 points to IORAM, single-word write, one wait-state          */
a0h=a0h+1   /* 1-cycle instruction, no EMI access -- wait-state is transparent  */
```

READY Pin

The READY input pin permits an external device to extend the length of an EMI access cycle. The READY pin can be used if the number of access cycles programmable in the **mwait** register (Table 58 on page 101) is insufficient, or if the desired number of access cycles varies from access to access. To use the READY pin for a memory segment access, the access time field in the **mwait** register (IATIM[3:0], YATIM[3:0], or XATIM[3:0]) must be programmed to a value of four or greater and the corresponding RDYEN[2:0] field of **mwait** must be set. If the access time field in **mwait** for the memory segment is less than four or if the RDYEN[2:0] field of **mwait** for the memory segment is cleared, then the DSP16210 ignores the READY pin when accessing that segment. On device reset, the RDYEN[2:0] fields are cleared, causing the DSP16210 to ignore the READY pin by default. Figure 39 on page 152 illustrates the operation of the READY pin.

The DSP16210 internally synchronizes the READY pin to the internal clock (CLK). READY must be asserted at least five cycles (plus a setup time[1]) prior to the end of the external memory operation. The DSP16210 adds the number of cycles that READY is asserted to the access time.

1. The READY pin setup time is t140 in Table 108 on page 152.

Hardware Architecture (continued)

Enhanced Serial I/O (ESIO) Unit

The ESIO is a programmable, hardware-managed, double-buffered, full-duplex serial I/O port designed to support glueless multichannel I/O processing on a TDM (time-division multiplex) highway. It has a 4-pin input interface (EIFS, EIBC, EDI, and EIBF) and a 5-pin output interface (EOFS, EOBC, EDO, EOEB, and EOBE). See Signal Descriptions beginning on page 121 for more details. ESIO input and output bit clocks are passive, i.e., must be provided by an external source. Data is transmitted and received in an LSB-first manner. The ESIO supports two modes of operation:

1. Simple mode: Serial I/O that has programmable 8-bit or 16-bit data lengths. The maximum serial data rate is 26 Mbits/s.

2. Frame mode: Up to 16 logical channels are multiplexed and demultiplexed on a standard 256-bit/frame TDM highway or on a 64-/128-/192-bit/frame highway[1]. The sample length for each channel is individually programmed as 1, 2, 4, or 8 bits corresponding to 8 Kbits/s, 16 Kbits/s, 32 Kbits/s, and 64 Kbits/s for a 2.048 Mbits/s TDM highway. The maximum supported serial data rate is 8.192 Mbits/s.

The ESIO communicates I/O buffer status to the core using the input buffer full (EIBF), output buffer empty (EOBE), input frame error (EIFE), output frame error (EOFE), and output collision (ECOL) interrupts. The input buffer full and output buffer empty conditions are also indicated via the EIBF and EOBE pins. In frame mode, EIBF and EOBE are based upon the completion of a programmable number of frames.

The ESIO contains 16 memory-mapped, double-buffered serial-to-parallel input demultiplexer registers (**IDMX⟨0—15⟩**). These 16-bit read-only registers can be configured to demultiplex a maximum of 16 logical input channels. A logical input channel is a nonoverlapping sequence of consecutive bits (1, 2, 4, or 8) identified by a starting bit position within the frame.

The ESIO also contains 16 memory-mapped, double-buffered parallel-to-serial output multiplexer registers (**OMX⟨0—15⟩**). These 16-bit write-only registers can be configured to multiplex a maximum of 16 logical output channels. A logical output channel is a nonoverlap-

ping sequence of consecutive bits (1, 2, 4, or 8) identified by a starting bit position within the frame.

The ESIO's serial data output (EDO) supports multi-master operation and can be configured as open-drain or 3-state.

Input Section

The control registers in the ESIO input section are the input control register (**ICR**), input channel start bit registers (**ICSB⟨0—7⟩**), input channel start length registers (**ICSL⟨0—1⟩**), and input channel valid vector register (**ICVV**). The data registers are **IDMX⟨0—15⟩**. All the ESIO input section registers are 16 bits and are memory-mapped as illustrated in Table 10.

Table 10. ESIO Memory Map (Input Section)

Memory Address	Register	R/W[†]	Memory Address	Register	R/W[†]
0xE0000	**IDMX0**	R	0xE0010	**ICSB0**	R/W
0xE0001	**IDMX1**	R	0xE0011	**ICSB1**	R/W
0xE0002	**IDMX2**	R	0xE0012	**ICSB2**	R/W
0xE0003	**IDMX3**	R	0xE0013	**ICSB3**	R/W
0xE0004	**IDMX4**	R	0xE0014	**ICSB4**	R/W
0xE0005	**IDMX5**	R	0xE0015	**ICSB5**	R/W
0xE0006	**IDMX6**	R	0xE0016	**ICSB6**	R/W
0xE0007	**IDMX7**	R	0xE0017	**ICSB7**	R/W
0xE0008	**IDMX8**	R	0xE0018	**ICSL0**	R/W
0xE0009	**IDMX9**	R	0xE0019	**ICSL1**	R/W
0xE000A	**IDMX10**	R	0xE001A	**ICR**	R/W
0xE000B	**IDMX11**	R	0xE001B	**ICVV**	R/W
0xE000C	**IDMX12**	R	0xE001C	RESERVED	
0xE000D	**IDMX13**	R	0xE001D		
0xE000E	**IDMX14**	R	0xE001E		
0xE000F	**IDMX15**	R	0xE001F		

† This column indicates whether the register is readable (R) and/or writable (W).

The input control register (**ICR**) (Table 47 on page 95) controls the configuration of the input section, including the selection of simple mode vs. frame mode. **ICVV** (Table 50 on page 96) specifies the number of active logical channels (one for simple mode and 1 through 16 for frame mode). **ICSB⟨0—7⟩** (Table 48 on page 96) and **ICSL⟨0—1⟩** (Table 49 on page 96) are used only in frame mode. They specify the starting bit position and the sample length (1, 2, 4, or 8 bits) of each logical channel.

---

1. A single DSP16210 can process up to 128 bits/frame (sixteen 8-bit channels).

Hardware Architecture (continued)

Enhanced Serial I/O Unit (ESIO) (continued)

ESIO interrupts are summarized in Table 14 on page 41.

Input Control Signal Conditioning. As illustrated in Figure 7, ILEV (bit 3) of the **ICR** register (Table 47 on page 95) selects the polarity of the input bit clock, EIBC. The modified clock is the input bit clock for the input section (IBC). The input sync level, ISLEV (bit 5) of the **ICR** register, selects whether or not the input frame sync, EIFS, is inverted. If input sync delay (ISDLY) (bit 6) of the **ICR** register is zero, this modified signal is the frame sync for the input section (IFS). If ISDLY is one, the modified signal is first retimed by the EIBC clock before becoming the frame sync for the input section (IFS). Figure 8 illustrates the timing of IFS when ISDLY is one.
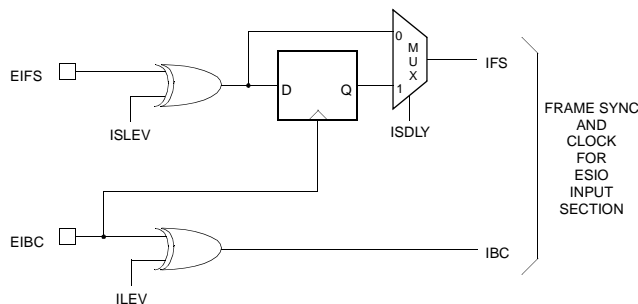
Figure 7. Input Control Signal Conditioning
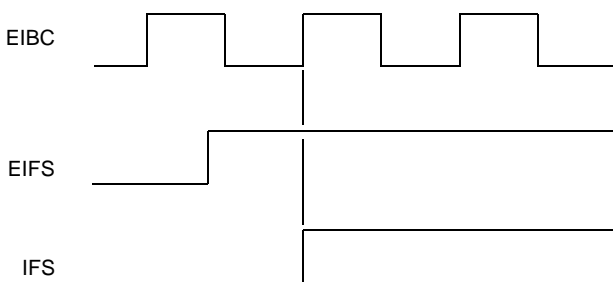
Figure 8. Frame Sync Timing with ILEV = ISLEV = 0 and ISDLY = 1

The rising edge of IFS (captured by the next rising edge of IBC) indicates that the first bit of the serial input packet or frame (from EDI) is captured by the falling edge of IBC. This edge also initializes the internal bit counter to zero, and every subsequent rising edge of IBC increments the bit counter. In frame mode, this bit counter is used by the input control hardware to define logical channel start points and to detect input frame errors. See Figure 9.
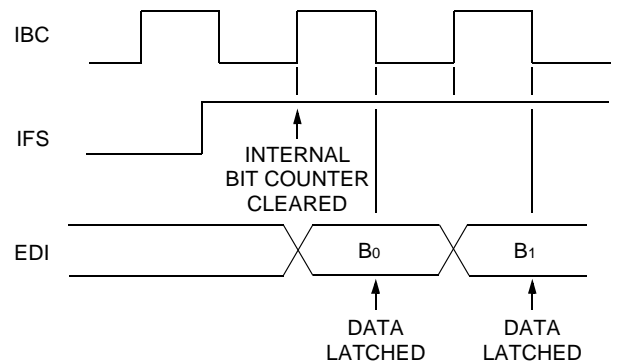
Figure 9. Input Functional Timing

Simple Input Mode Processing. The ESIO input block operates in simple input mode when IMODE (bit 8) of the **ICR** register is set to 1. In this mode, the programmer must set the **ICVV** register to 0x0001. The ESIO disables the input frame error interrupt (EIFE).

In simple mode, the ESIO supports double-buffered 8-bit and 16-bit LSB-first serial operation. Eight-bit serial operation is selected by setting ISIZE (bit 7) of the **ICR** register. This right justifies 8-bit input packets, i.e., the 8-bit data is aligned with bits [7:0] of **IDMX0**.

See Figure 10 on page 34 for a diagram of the input demultiplexer structure. Serial input data from EDI is captured into a serial-to-parallel register by the falling edge of IBC (illustrated as IBCQ0 in Figure 10). After all programmed bits (8 or 16) have been captured, the data is transferred to the parallel data register **IDMX0** for future core processing (for example, $a0h = *r0$, where **r0** points to location 0xE0000).

The ESIO asserts the input buffer full (EIBF) output pin and the EIBF interrupt after the falling edge of the final IBC capture clock. The EIBF interrupt and pin are cleared when the DSP reads the **IDMX0** memory-mapped register. EIBF is also cleared on device reset or if the DSP program resets the input section (writes **ICR** with the IRESET field (bit 4) set). The simple mode input timing diagram (for ILEV = 0, ISLEV = 0, ISIZE = 0, and ISDLY = 0) is illustrated in Figure 52 on page 166.

Hardware Architecture (continued)

Enhanced Serial I/O Unit (ESIO) (continued)

Figure 10. Input Demultiplexer (IDMX) and Register File Structure

Hardware Architecture (continued)

Enhanced Serial I/O Unit (ESIO) (continued)

Frame Input Mode Processing. The ESIO operates in frame input mode (IMODE) when (bit 8) of the **ICR** register is cleared. (IMODE is cleared on reset.) The ESIO demultiplexes multiple channels from a serial stream consisting of a frame of 64, 128, 192, or 256 bits. The input frame size (IFRMSZ) is specified by (bits [11:10]) of the **ICR** register. The start of a new frame is signaled by the rising edge of the input frame sync (IFS). The ESIO ignores input until it has detected the beginning of a valid frame. Serial data is captured by the falling edge of the input bit clock (IBC) (see Figure 9 on page 33).

See Figure 10 on page 34 for a diagram of the **IDMX** structure. The input section contains 16 double-buffered 16-bit serial-to-parallel input demultiplexers with provision for either 8-bit (ISIZE = 1) or 16-bit (ISIZE = 0) right-justified data. Each logical channel has a dedicated 16-bit shift register that receives demultiplexed serial data and has a dedicated 16-bit parallel read register (**IDMX⟨0—15⟩**). Each shift register is clocked individually by IBCQ[15:0], a qualified IBC bit clock that starts when the internal bit counter matches the input logical channel start bit specified by the corresponding **ICSB⟨0—7⟩** register (see Table 11).

Table 11. Input Channel Start Bit Registers

	15—8	7—0
ICSB0	Channel 1	Channel 0
ICSB1	Channel 3	Channel 2
ICSB2	Channel 5	Channel 4
ICSB3	Channel 7	Channel 6
ICSB4	Channel 9	Channel 8
ICSB5	Channel 11	Channel 10
ICSB6	Channel 13	Channel 12
ICSB7	Channel 15	Channel 14

Field	Value	Description
Channel 0 to Channel 15	0x00 to 0xFF	Start bit position for corresponding logical input channel. Ranges from 0 to 255.

The clock IBCQ[15:0] is asserted in each frame for the number of cycles that matches the programmed sample length for the corresponding logical channel. The sample length is specified by one of the **ICSL⟨0—1⟩** registers (see Table 49 on page 96).

Figure 11 is a timing diagram that depicts the clock IBCQ3 assuming that the sample length is 2 bits (**ICSL0**[7:6] = 01) and the start bit is 63 (**ICSB1**[15:8] = 0x3F). In this example, bits B_{63} and B_{64} are clocked into the shift register for logical channel 3.

When 16 serial input bits have been captured for a given channel n, the ESIO asserts IDLDn (see Figure 10 on page 34), transferring the shift register contents to the channel's parallel read register, **IDMXn**. This transfer occurs every 2, 4, 8, or 16 frames depending on the sample length programmed for channel n via **ICSL⟨0—1⟩**. This serial-to-parallel transfer permits a 16-bit word of channel data to be captured (at the IBC rate) while the previous word is read by the core.
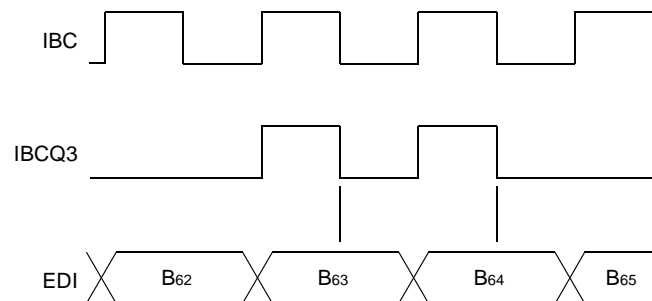
Figure 11. Serial Input Clocking Example

The ESIO is programmed to generate the input buffer full (EIBF) interrupt and assert the EIBF output pin at the completion of every 2, 4, 8, or 16 frames depending on the IFIR field of the **ICR** register (Table 47 on page 95). EIBF is cleared if the DSP program reads any of the **IDMX⟨0—15⟩** registers or if it resets the input section (writes **ICR** with the IRESET field (bit 4) set). EIBF is first asserted when the programmed number of input frames have been received following initialization of the ESIO input section. The programmer initializes the input section by simultaneously resetting it and enabling it, i.e., by writing **ICR** with the IRESET field set and the ICA field (bit 2) set. The IRESET field clears itself automatically every cycle of the internal clock (CLK). Therefore, when **ICR** is read, the value of the IRESET field is always clear.

Hardware Architecture (continued)

Enhanced Serial I/O Unit (ESIO) (continued)

Prior to initializing the input section as described above, the programmer must configure **ICVV**, **ICSB**⟨**0—7**⟩, and **ICSL**⟨**0—1**⟩. The write of **ICR** that initializes the input section must also configure the input section appropriately (IMODE, IFRMSZ, etc.). Before changing any input channel attributes (e.g., **ICVV**, IMODE), the programmer must first reset the input section. Specifically, the programmer must write **ICR** with the IRESET field (bit 4) set and the ICA field (bit 2) clear, change the attributes, and then enable the input section by writing **ICR** with the ICA field (bit 2) set.

In an environment with several different logical channel sampling lengths, the EIBF generation rate should be set to the highest serial-to-parallel transfer rate (see Table 49 on page 96). Each channel is serviced at its programmed rate when a full word of input data is provided. For example, in a system with logical channels of sample length 1, 2, and 8 bits, the highest serial-to-parallel transfer rate is every two frames and IFIR should be programmed to 0 (one EIBF every two frames). The channels with an 8-bit sample length should be serviced every EIBF interrupt, the channels with a 2-bit sample length should be serviced every four EIBF interrupts, and the channels with a 1-bit sample length should be serviced every eight EIBF interrupts.

The ITMODE field (bit 9) of **ICR** can override the serial-to-parallel transfer rate specified by **ICSL**⟨**0—1**⟩. When ITMODE is set to 1, data is transferred from each input shift register to all sixteen **IDMX**⟨**0—15**⟩ registers simultaneously at the programmed IFIR frequency. The ESIO asserts EIBF for each transfer. It is not necessary to reset the ESIO prior to changing the ITMODE bit.

Note: In ITMODE, input data is not necessarily right-justified in **IDMX**⟨**0—15**⟩. The LSB of the data stream is continuously shifted into the MSB (8-bit or 16-bit) location of each shift register. The contents of the shift registers are transferred to the **IDMX**⟨**0—15**⟩ registers at the alternate IFIR frequency.

The logical channels are enabled by programming the 16-bit **ICVV** register. Each bit in this register corresponds to a logical channel, e.g., bit 5 of **ICVV** corresponds to logical channel 5. When a bit in **ICVV** is set, the ESIO demultiplexes the input data stream for the corresponding channel. The bits in **ICVV** must be packed, i.e., channels must be allocated from 0 to 15 with no holes between valid channels. For example, if

ICVV contains 0x00FF, then logical channels 0—7 are enabled and demultiplexed. A value of 0x08FF for **ICVV** is invalid because the channels are not packed.

Logical channels must be assigned in increasing input channel start bit order and must not overlap. For example, if channel 4 has a start bit of 48 and a sample length of 4 bits (**ICSB2**[7:0] = 0x30; **ICSL0**[9:8] = 10), then channel 5 must have a start bit value greater than or equal to 52 (48 + 4).

The ESIO reports an input frame error (EIFE) when it is processing a valid frame and an input frame sync is detected before the number of bits in the programmed frame length (IFRMSZ in **ICR**) have been sampled. If an EIFE interrupt occurs, the DSP program should reset the input section by writing **ICR** with the IRESET bit set.

Output Section

The control registers in the ESIO output section are the output control register (**OCR**), the output channel start bit registers (**OCSB**⟨**0—7**⟩), the output channel sample length registers (**OCSL**⟨**0—1**⟩), and the output channel valid vector register (**OCVV**). The data registers are **OMX**⟨**0—15**⟩. All the ESIO output section registers are 16 bits and are memory mapped as illustrated in Table 12.

Table 12. ESIO Memory Map (Output Section)

Memory Address	Register	R/W†	Memory Address	Register	R/W†
0xE0020	**OMX0**	W	0xE0030	**OCSB0**	R/W
0xE0021	**OMX1**	W	0xE0031	**OCSB1**	R/W
0xE0022	**OMX2**	W	0xE0032	**OCSB2**	R/W
0xE0023	**OMX3**	W	0xE0033	**OCSB3**	R/W
0xE0024	**OMX4**	W	0xE0034	**OCSB4**	R/W
0xE0025	**OMX5**	W	0xE0035	**OCSB5**	R/W
0xE0026	**OMX6**	W	0xE0036	**OCSB6**	R/W
0xE0027	**OMX7**	W	0xE0037	**OCSB7**	R/W
0xE0028	**OMX8**	W	0xE0038	**OCSL0**	R/W
0xE0029	**OMX9**	W	0xE0039	**OCSL1**	R/W
0xE002A	**OMX10**	W	0xE003A	**OCR**	R/W
0xE002B	**OMX11**	W	0xE003B	**OCVV**	R/W
0xE002C	**OMX12**	W	0xE003C	RESERVED	
0xE002D	**OMX13**	W	0xE003D		
0xE002E	**OMX14**	W	0xE003E		
0xE002F	**OMX15**	W	0xE003F		

† This column indicates whether the register is readable (R) and/or writable (W).

Hardware Architecture (continued)

Enhanced Serial I/O Unit (ESIO) (continued)

OCR (Table 59 on page 102) controls the configuration of the output section, including the selection of simple mode vs. frame mode. **OCVV** (Table 62 on page 103) specifies the number of active logical channels (one for simple mode and 1 through 16 for frame mode). **OCSB⟨0—7⟩** (Table 60 on page 103) and **OCSL⟨0—1⟩** (Table 61 on page 103) are used only in frame mode. They specify the starting bit position and the sample length (1, 2, 4, or 8 bits) of each logical channel.

As illustrated in Figure 12, OLEV (bit 3) of the **OCR** register selects the polarity of the output bit clock, EOBC. This modified clock is the output bit clock for the output section (OBC). OSLEV (bit 8) of the **OCR** register selects whether or not the output frame sync, EOFS, is inverted. This modified signal is the frame sync for the output section (OFS).
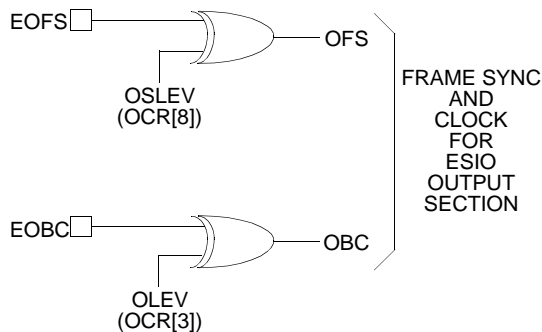
Figure 12. Output Control Signal Conditioning

As illustrated in Figure 13, the ESIO drives serial data onto the ESIO data out (EDO) pin the rising edge of the output bit clock (OBC). The rising edge of output frame sync (OFS) indicates that the first bit of the serial output packet or frame is driven onto EDO on the next rising edge of OBC. This edge (as captured by OBC) also initializes the internal bit counter to zero, and every subsequent rising edge of OBC increments the bit counter. In frame mode, this bit counter is used by the output control hardware to define logical channel start points and to detect output frame errors.

The ESIO asserts the EOBE output pin and the EOBE interrupt on the falling edge of OBC following detection of OFS as shown in Figure 13. EOBE is cleared when the DSP program writes any of the **OMX⟨0—15⟩** memory-mapped registers. EOBE is also cleared on device reset or if the DSP program resets the output section by writing the **OCR** register with the ORESET field (bit 4) set.
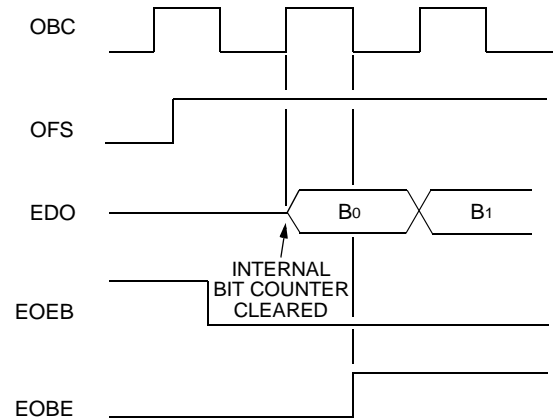
Figure 13. Output Functional Timing

The ESIO drives EDO only during its scheduled timeslot as illustrated in Figure 13. Otherwise EDO is in the high-impedance state. The other necessary conditions for the DSP16210 to drive EDO are:

■ The EOEB negative-assertion input pin must be asserted (low).

■ The EDOEO bit in the **OCR** register (bit 6) must be set.

If EOEB is high or if the EDOEO bit is cleared, then EDO is in the high-impedance state regardless of the state of ESIO output section. The EDOEO bit is cleared on reset causing the EDO pin to be in the high-impedance state by default.

The EDOMD bit in **OCR** (bit 5) configures the EDO output pin driver as either 3-state or open-drain.

Hardware Architecture (continued)

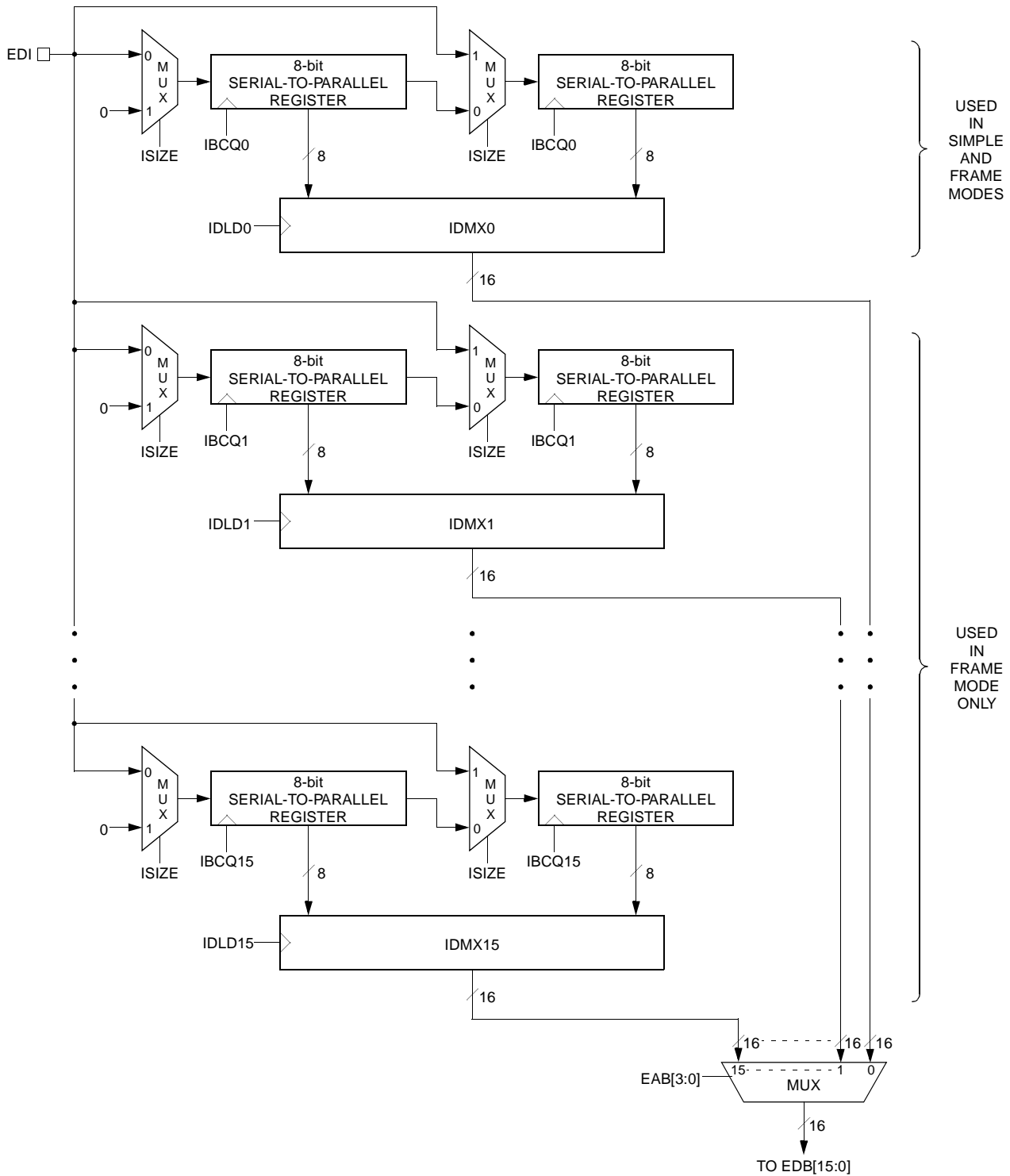### Enhanced Serial I/O Unit (ESIO) (continued)
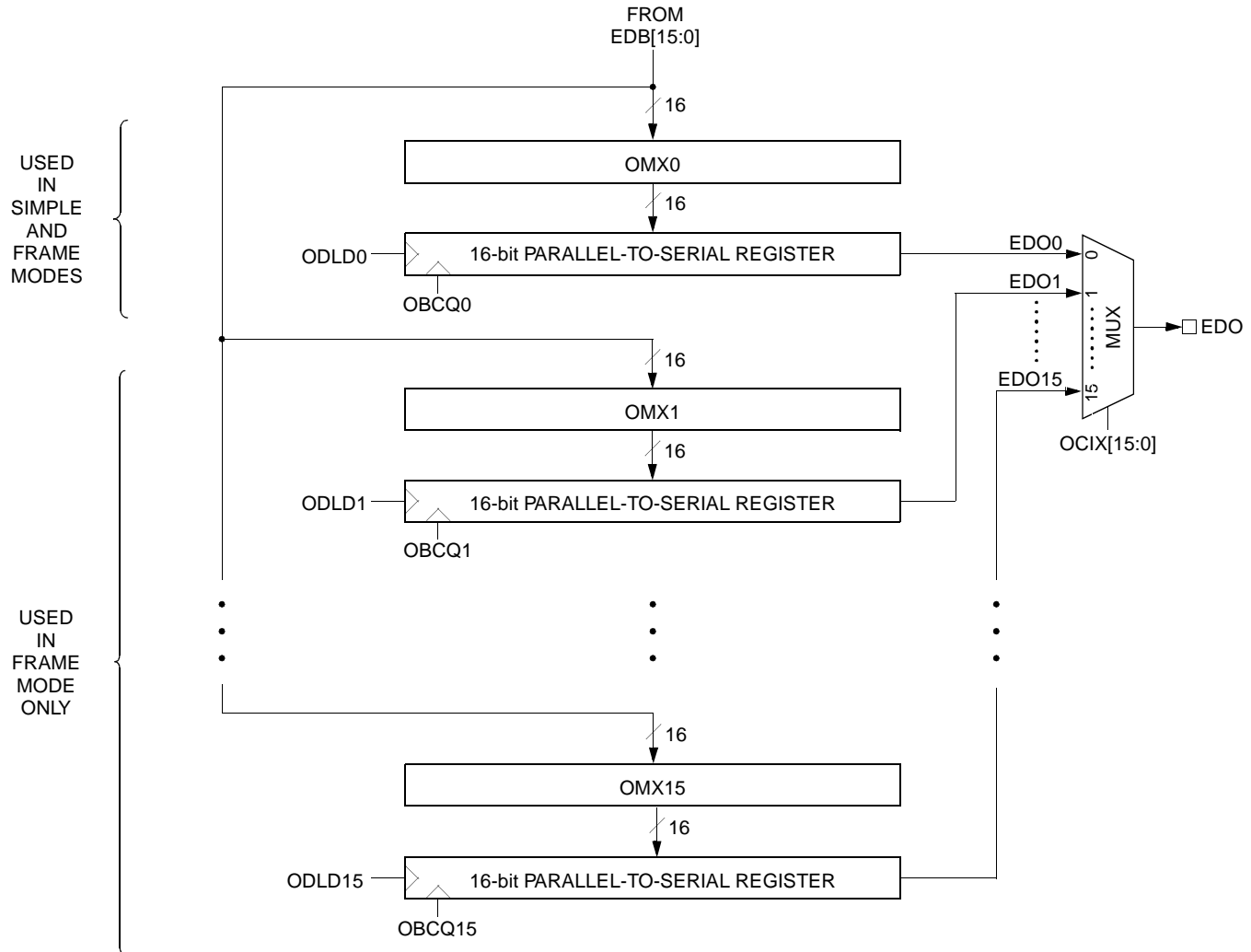
Figure 14. Output Multiplexer (OMX) and Register File Structure

Hardware Architecture (continued)

Enhanced Serial I/O Unit (ESIO) (continued)

Simple Output Mode Processing. The ESIO output block operates in simple mode when OMODE (bit 10) of the **OCR** register is set to 1. In this mode, the programmer must set the **OCVV** register to 0x0001. The ESIO disables the output frame error (EOFE) and output collision (ECOL) interrupts.

In simple mode, the ESIO supports double-buffered 8-bit and 16-bit LSB-first serial operation. Eight-bit serial operation is selected by setting OSIZE (bit 9) of the **OCR** register.

See Figure 14 on page 38 for a diagram of the output multiplexer structure. The program writes 8-bit or 16-bit data into the **OMX0** register. (8-bit data must be right-justified in **OMX0**). On the rising edge of the first OBC clock after frame sync (OFS) detection, the data is transferred from **OMX0** to the 16-bit parallel-to-serial register (ODLD0 in Figure 14 is asserted). During this same OBC clock (illustrated as OBCQ0 in Figure 14), the LSB of the data (B_0) is applied to the EDO pin. On each subsequent rising edge of OBC, the remaining bits are applied to EDO. The simple mode output timing diagram (for OLEV = 0, OSLEV = 0, and OSIZE = 0) is illustrated in Figure 53 on page 167.

Frame Output Mode Processing. The ESIO operates in frame output mode when OMODE (bit 10) of the **OCR** register is cleared. (OMODE is cleared on reset.) The ESIO multiplexes up to 16 channels of data onto a serial stream consisting of a frame of 64, 128, 192, or 256 bits. The frame size is specified by OFRMSZ (bits [13:12]) of the **OCR** register. The start of a new frame is signaled by the rising edge of the output frame sync (OFS). Serial data is captured by the falling edge of the output bit clock (OBC) (see Figure 13 on page 37).

See Figure 14 on page 38 for a diagram of the output multiplexed into EDO. The output section contains 16 double-buffered 16-bit parallel-to-serial output multiplexers. Each logical channel has a dedicated 16-bit parallel write register (**OMX⟨0—15⟩**) and has a dedicated 16-bit shift register that transmits serial data for that channel. Each shift register is clocked individually by OBCQ[15:0], a qualified OBC bit clock that starts when the internal bit counter matches the output logical channel start bit specified by the corresponding **OCSB⟨0—7⟩** register (see Table 13).

Table 13. Output Channel Start Bit Registers

	15—8	7—0
OCSB0	Channel 1	Channel 0
OCSB1	Channel 3	Channel 2
OCSB2	Channel 5	Channel 4
OCSB3	Channel 7	Channel 6
OCSB4	Channel 9	Channel 8
OCSB5	Channel 11	Channel 10
OCSB6	Channel 13	Channel 12
OCSB7	Channel 15	Channel 14

Field	Value	Description
Channel 0 to Channel 15	0x00 to 0xFF	Start bit position for corresponding logical output channel. Ranges from 0 to 255.

OMX⟨0—15⟩ are 16-bit write-only memory-mapped registers that are written if the core writes to the corresponding memory location (see Table 12 on page 36). The ESIO asserts ODLD*n* for logical channel *n* (see Figure 14 on page 38) to load the channel's parallel-to-serial register with the contents of the **OMX*n*** register. All 16 parallel-to-serial registers are loaded simultaneously when the first frame sync (OFS) is asserted following initialization of the output section. (See the following discussion for a description of output section initialization.) The parallel-to-serial register for channel *n* is subsequently loaded (ODLD*n* asserted) every 2, 4, 8, or 16 frames depending on the sample length programmed for channel *n* via **OCSL⟨0—1⟩** (see Table 61 on page 103). This transfer permits the core to write a new 16-bit word of channel data into **OMX*n*** while the old word is shifted out serially.

Hardware Architecture (continued)

Enhanced Serial I/O Unit (ESIO) (continued)

The clock OBCQ[15:0] is asserted in each frame for the number of cycles that matches the programmed sample length for the corresponding logical channel. The sample length is specified by one of the **OCSL⟨0—1⟩** registers (see Table 61 on page 103).

Figure 15 is a timing diagram of the clock OBCQ3, the bit clock for logical channel 3, assuming that the sample length is 2 bits (**OCSL0**[7:6] = 01) and the start bit is 63 (**OCSB1**[15:8] = 0x3F). In Figure 15, D_0 is the LSB and the initial output of the channel 3 parallel-to-serial register (EDO3). The ESIO asserts the OCIX3 signal during the time slot for logical channel 3, enabling EDO3 onto the EDO pin (see Figure 14 on page 38). OBCQ3 is asserted for two cycles, shifting the parallel-to-serial register contents by two bit positions, leaving D_2 on EDO3 for the next frame.
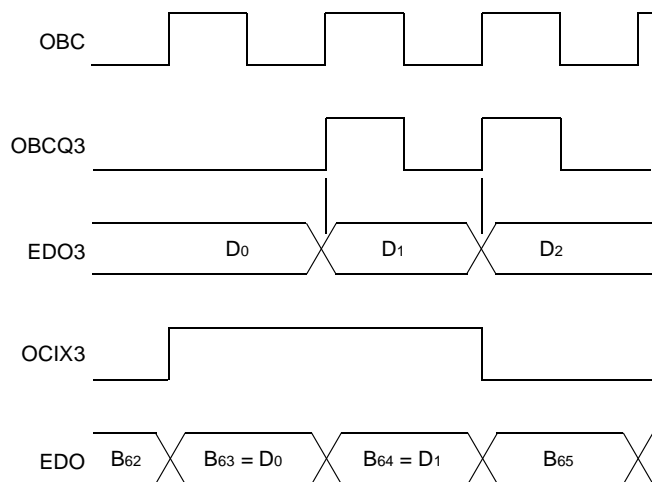
Figure 15. Serial Output Clocking Example

The ESIO asserts the EOBE output pin and the EOBE interrupt on the falling edge of OBC after the detection of the first OFS following output section initialization. (See Figure 13 on page 37 for an illustration of EOBE timing and the discussion below for a description of output section initialization.) EOBE is cleared when the DSP program writes any of the **OMX⟨0—15⟩** memory-mapped registers. EOBE is also cleared on device reset or if the DSP program resets the output section by writing the **OCR** register with the ORESET field (bit 4) set. The ESIO reasserts EOBE at the completion of every 2, 4, 8, or 16 frames depending on the OFIR field (bits [1:0]) of the **OCR** register (Table 59 on page 102).

The programmer initializes the output section by simultaneously resetting it and enabling it, i.e., by writing **OCR** with the ORESET field set and the OCA field (bit 2) set. The ORESET field clears itself automatically every cycle of the internal clock (CLK). Therefore, when **OCR** is read, the value of the ORESET field is clear.

Prior to initializing the output section as described above, the programmer must configure **OCVV**, **OCSB⟨0—7⟩**, and **OCSL⟨0—1⟩**. The write of **OCR** that initializes the output section must also configure the output section appropriately (OMODE, OFRMSZ, etc.). Before changing any output channel attributes (e.g., **OCVV**, OMODE), the programmer must first reset the output section. Specifically, the programmer must write **OCR** with the ORESET field (bit 4) set and the OCA field (bit 2) clear, change the attributes, and then enable the output section by writing **OCR** with the OCA field (bit 2) set.

In an environment with several different logical channel sampling lengths, the EOBE generation rate should be set to the highest parallel-to-serial transfer rate (see Table 61 on page 103). Each channel is serviced at its programmed rate when a full word of output data has been transmitted. For example, in a system with logical channels of sample length 1, 2, and 8 bits, the highest parallel-to-serial transfer rate is every 2 frames and OFIR should be programmed to 0 (one EOBE every 2 frames). The channels with an 8-bit sample length should be serviced every EOBE interrupt, the channels with a 2-bit sample length should be serviced every four EOBE interrupts, and the channels with a 1-bit sample length should be serviced every eight EOBE interrupts.

The OTMODE field (bit 11) of **OCR** can override the parallel-to-serial transfer rate specified by **OCSL⟨0—1⟩**. When OTMODE is set to 1, data is transferred from each **OMX⟨0—15⟩** register to all 16 output shift registers simultaneously at the programmed OFIR frequency.

The logical channels are enabled by programming the 16-bit **OCVV** register. Each bit in this register corresponds to a logical channel, e.g., bit 5 of **OCVV** corresponds to logical channel 5. When a bit in **OCVV** is set, the ESIO multiplexes the output serial stream with data from the corresponding channel. The bits in **OCVV** must be packed, i.e., channels must be allocated from 0 to 15 with no holes between valid channels. For example, if **OCVV** contains 0x00FF, then logical channels 0—7 are enabled and multiplexed. A value of 0x08FF for **OCVV** is invalid because the channels are not packed.

Hardware Architecture (continued)

Enhanced Serial I/O Unit (ESIO) (continued)

Logical channels must be assigned in increasing output channel start bit order and must not overlap. For example, if channel 4 has a start bit of 48 and a sample length of 4 bits (**OCSB2**[7:0] = 0x30; **OCSL0**[9:8] = 10), then channel 5 must have a start bit value greater than or equal to 52 (48 + 4).

The ESIO reports an output frame error (EOFE) when it is processing a valid frame and an output frame sync is detected before the number of bits in the programmed frame length (OFRMSZ in **OCR**) have been transmitted. If an EOFE interrupt occurs, the DSP program should reset the output section by writing **OCR** with the ORESET bit set.

When driving output data in frame mode with EDO programmed as an open-drain device (EDOMD = 1 and EDOEO = 1), the ESIO samples the EDO pin every EOBC clock cycle. If the sampled value is not the intended output value, the ESIO has collided with another serial bus agent. When a bus collision is detected, the ESIO asserts the output collision interrupt (ECOL). The DSP program clears ECOL by writing **OCR** with either the ORESET field (bit 4) or the CRESET field (bit 7) set.

Table 14 summarizes the ESIO interrupts. See for information on request clearing latency for these interrupts.

Table 14. ESIO Interrupts

Interrupt	Name	Description		Cleared By
EIBF	Input Buffer Full	Simple Mode (IMODE = 1)	Asserted if a programmed number of input bits (8 or 16 depending on ISIZE (**ICR**[7])) have been captured following assertion of the input frame sync.	Any of the following: ■ Device reset. ■ DSP program reads any of **IDMX⟨0—15⟩**. ■ The DSP program sets the IRESET field (**ICR**[4]).
		Frame Mode (IMODE = 0)	Asserted after N input frames (N = 2, 4, 8, or 16, depending on IFIR[1:0] (**ICR**[1:0])) have been received following input section initialization[†]. If ICA (**ICR**[2]) remains set, EIBF is reasserted after every subsequent N frames have been received.	
EIFE[‡]	Input Frame Error	Frame Mode (IMODE = 0)	Asserted if the input section is processing a valid frame and an input frame sync is detected before the number of bits specified by IFRMSZ[1:0] (**ICR**[11:10]) have been sampled.	The DSP program sets the IRESET field (**ICR**[4]).
EOBE	Output Buffer Empty	Simple Mode (OMODE = 1)	Asserted after the first bit (LSB) has been output.	Any of the following: ■ Device reset. ■ DSP program writes any of **OMX⟨0—15⟩**. ■ The DSP program sets the ORESET field (**OCR**[4]).
		Frame Mode (OMODE = 0)	Asserted after the first bit (LSB) of the first frame has been output following output section initialization[§]. If OCA (**OCR**[2]) remains set, EOBE is reasserted at the completion of every 2, 4, 8, or 16 frames depending on OFIR[1:0] (**OCR**[1:0]).	
EOFE[‡]	Output Frame Error	Frame Mode (OMODE = 0)	Asserted if the output section is processing a valid frame and an output frame sync is detected before the number of bits in the programmed frame length OFRMSZ[1:0] (**OCR**[13:12]) have been transmitted.	The DSP program sets the IRESET field.
ECOL[‡]	Output Collision	Frame Mode (OMODE = 0)	Asserted if EDO is an open-drain output (EDOMD (**OCR**[5]) = 1 and EDOEO (**OCR**[6]) = 1) and the sampled EDO pin value is not the intended output value.	The DSP program sets the ORESET field (**OCR**[4]) or the CRESET field (**OCR**[7]).

† The DSP program initializes the input section by setting IRESET (**ICR**[4]) and ICA (**ICR**[2]).
‡ This interrupt is disabled in simple mode.
§ The DSP program initializes the output section by setting ORESET (**OCR**[4]) and OCA (**OCR**[2]).

Hardware Architecture (continued)

Modular I/O Units (MIOUs)

The DSP16210 contains two identical modular I/O units: MIOU0 (provides DMA for the PHIF16) and MIOU1 (provides DMA for the SSIO).

An MIOU provides programmable DMA capability. Figure 16 shows the MIOUs, their connections to the IORAMs, the attached I/O peripherals, and the IDB. Each MIOU interfaces its attached peripheral to a single 1 Kword bank of IORAM storage that resides in the DSP16000 core's Y-memory space. Input and output buffers for each peripheral are allocated in each IORAM.
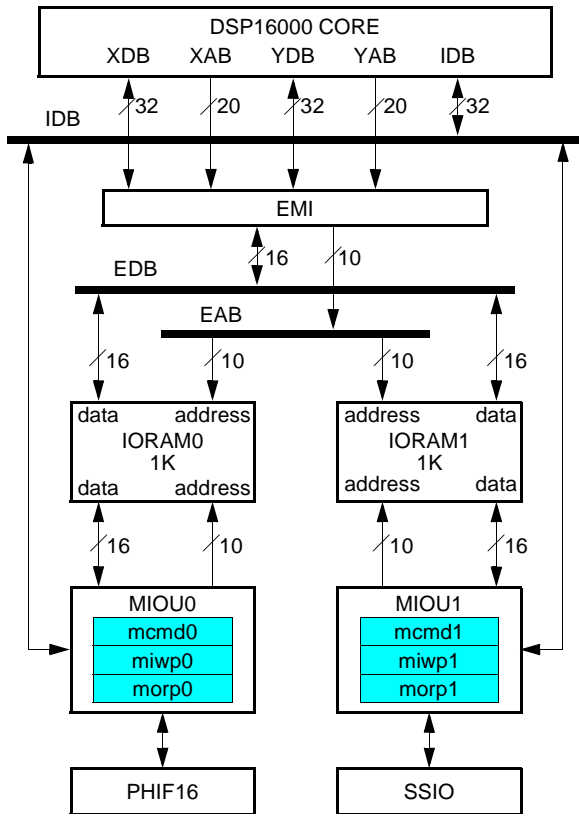
Figure 16. Modular I/O Units

IORAM

IORAM storage consists of two 1 Kword banks of memory, IORAM0 and IORAM1. Each IORAM bank has two 16-bit data and two 10-bit address ports. An IORAM bank can be shared by the core and an MIOU to implement a DMA-based I/O system. IORAM supports concurrent core execution and MIOU I/O processing.

Portions of IORAM not dedicated to I/O processing can be used as general-purpose data storage. However, a high collision rate between core and MIOU accesses to IORAM impacts core and I/O performance.

The IORAMs reside in the core's Y-memory space (see Figure 6 on page 26). The EMI interfaces the core to the IORAMs by translating between YAB/YDB accesses and EAB/EDB accesses. This translation is functionally transparent to the programmer. The core can access the IORAM as single words or as double words and the EMI automatically performs the required multiplexing and sequencing. Core accesses to IORAM cause the core to incur wait-states (see External Memory Interface (EMI) beginning on page 27). If the core and an MIOU simultaneously access the same IORAM, the MIOU access occurs first followed by the core access and the core incurs a conflict wait-state.

MIOU Registers

For each MIOU, software controls DMA operations by programming three registers that are directly program-accessible: **mcmd⟨0,1⟩**, **miwp⟨0,1⟩**, and **morp⟨0,1⟩**. See Table 16 on page 43 for a description of these registers.

In the DSP16000 instruction set, **mcmd⟨0,1⟩**, **miwp⟨0,1⟩**, and **morp⟨0,1⟩** are off-core registers in the RAB and RB register sets. Table 15 summarizes the instructions for programming these registers.

Table 17 on page 43 summarizes the MIOU registers that are accessible by executing an MIOU command. Software executes an MIOU command by writing to **mcmd⟨0,1⟩**. See MIOU Commands beginning on page 43 for more information.

Table 15. Instructions for Programming MIOU Registers

Instruction Syntax		Substitution		Example
RAB = IM20	RAB	**mcmd⟨0,1⟩**, **miwp⟨0,1⟩**, or **morp⟨0,1⟩**		`mcmd0 = 0x6000`
RB = aTE⟨h, l⟩	RB	**mcmd⟨0,1⟩**, **miwp⟨0,1⟩**, or **morp⟨0,1⟩**		`miwp1 = a3h`
aTE⟨h, l⟩ = RB				`a0l = morp0`

Hardware Architecture (continued)

Modular I/O Units (MIOUs) (continued)

MIOU Registers (continued)

Table 16. MIOU⟨0,1⟩ 16-Bit Directly Program-Accessible Registers

Register	Function	Encoding
mcmd⟨0,1⟩ (Write Only)	**MIOU⟨0,1⟩ Command Register**. Instructions write commands to this register to control the MIOU⟨0,1⟩ state and to configure other write-only registers. These other registers are the attached peripheral's control register (**PHIFC** or **SSIOC**) and the MIOU's internal command-accessible registers (see Table 17).	15—12 : Opcode, 11—0 : Parameter The 4-bit opcode specifies the command to be executed. The 12-bit parameter is data used by the command.
miwp⟨0,1⟩ (Read/Write)	**MIOU⟨0,1⟩ Input Write Pointer.** Contains the address of the IORAM⟨0,1⟩ location to which the attached peripheral will write its next input sample[†]. After the sample is written, the MIOU⟨0,1⟩ increments **miwp⟨0,1⟩**.	15—10 : Reserved, 9—0 : Input Write Pointer (IORAM⟨0,1⟩ Address)
morp⟨0,1⟩ (Read/Write)	**MIOU⟨0,1⟩ Output Read Pointer.** Contains the address of the IORAM⟨0,1⟩ location from which the attached peripheral will read its next output sample[†]. After the sample is read, the MIOU⟨0,1⟩ increments **morp⟨0,1⟩**.	15—10 : Reserved, 9—0 : Output Read Pointer (IORAM⟨0,1⟩ Address)

† Regardless of the size of the sample within the peripheral (8-bit or 16-bit), each sample uses one 16-bit IORAM location and is right-justified. The attached peripheral places each 8-bit input sample into the least significant byte of the 16-bit IORAM location and reads each 8-bit output sample from the least significant byte of the 16-bit IORAM location.

Table 17. MIOU Write-Only Command-Accessible Registers

Block	Register	Description	Size (bits)	Block	Register	Description	Size (bits)
MIOU0	**PHIFC**	PHIF16 control	12	MIOU1	**SSIOC**	SSIO control	12
	IBAS0	Input buffer base address	10		**IBAS1**	Input buffer base address	10
	ILIM0	Input buffer limit address	10		**ILIM1**	Input buffer limit address	10
	ILEN0	Input length counter[†]	12		**ILEN1**	Input length counter[†]	12
	OBAS0	Output buffer base address	10		**OBAS1**	Output buffer base address	10
	OLIM0	Output buffer limit address	10		**OLIM1**	Output buffer limit address	10
	OLEN0	Output buffer length	11		**OLEN1**	Output buffer length	11

† **ILEN0** and **ILEN1** are signed registers in two's complement format.

MIOU Commands

Table 18 on page 44 describes the encoding of **mcmd⟨0,1⟩**. Software executes an MIOU⟨0,1⟩ command by writing to **mcmd⟨0,1⟩**. A command consists of a 4-bit opcode and a 12-bit parameter. See the code segment examples below:

```
mcmd0 = 0x0155    /* Load IBAS0 with IORAM0 address 0x155                */
mcmd1 = 0x6000    /* Reset MIOU1                                         */
```

Hardware Architecture (continued)

Modular I/O Units (MIOUs) (continued)

MIOU Commands (continued)

Table 18. MIOU⟨0,1⟩ Command (mcmd⟨0,1⟩) Register

15—12	11—0
Opcode[3:0]	Parameter[11:0]

Opcode[3:0]	Parameter[11:0]		Command	Command Mnemonic	Action
0x0	10-bit IORAM input buffer base address.	0x*NNN*†	Load **IBAS⟨0,1⟩**	IBAS⟨0,1⟩_LD	**IBAS⟨0,1⟩** ← 0x*NNN*
0x1	10-bit IORAM input buffer limit address.	0x*NNN*†	Load **ILIM⟨0,1⟩**	ILIM⟨0,1⟩_LD	**ILIM⟨0,1⟩** ← 0x*NNN*
0x2	10-bit IORAM output buffer base address.	0x*NNN*†	Load **OBAS⟨0,1⟩**	OBAS⟨0,1⟩_LD	**OBAS⟨0,1⟩** ← 0x*NNN*
0x3	10-bit IORAM output buffer limit address.	0x*NNN*†	Load **OLIM⟨0,1⟩**	OLIM⟨0,1⟩_LD	**OLIM⟨0,1⟩** ← 0x*NNN*
0x4	11-bit unsigned input length update amount.	0x*NNN*‡	Update **ILEN⟨0,1⟩**	ILEN⟨0,1⟩_UP	**ILEN⟨0,1⟩** ← **ILEN⟨0,1⟩** + 0x*NNN* Activate§ peripheral service in MIOU⟨0,1⟩.
0x5	11-bit unsigned output length update amount.	0x*NNN*‡	Update **OLEN⟨0,1⟩**	OLEN⟨0,1⟩_UP	**OLEN⟨0,1⟩** ← **OLEN⟨0,1⟩** + 0x*NNN*
0x6	Must be zero.	0x000	Reset MIOU⟨0,1⟩	RESET⟨0,1⟩	Initialize MIOU⟨0,1⟩ control state and deactivate†† MIOU⟨0,1⟩ peripheral service. See Table 19 for the effect of reset on MIOU⟨0,1⟩ interrupts and registers.
0x7	12-bit value for peripheral control register (**PHIFC** or **SSIOC**‡‡).	0x*NNN*	Load Peripheral Control	PCTL⟨0,1⟩_LD	**PHIFC** ← 0x*NNN* (for MIOU0) or **SSIOC** ← 0x*NNN* (for MIOU1)
0x8	Must be zero.	0x000	Input Disable	INPT⟨0,1⟩_DS	Disable MIOU⟨0,1⟩ input processing. (Input processing is re-enabled by executing a subsequent ILEN⟨0,1⟩_UP command.)
0x9—0xF	Reserved.				

† 0x*NNN* is a 12-bit number for which the ten least significant bits (bits [9:0]) are an IORAM⟨0,1⟩ address and the two most significant bits (bits [11:10]) must be 0.
‡ 0x*NNN* is a 12-bit unsigned number for which the most significant bit (bit 11) must be 0.
§ Or reactivate peripheral service in MIOU⟨0,1⟩ if it has been deactivated by a prior RESET⟨0,1⟩ command.
†† Subsequent execution of an ILEN_UP⟨0,1⟩ command reactivates MIOU⟨0,1⟩ peripheral service.
‡‡ See Table 63 on page 104 and Table 70 on page 110.

Table 19. Effect of Reset on MIOU Interrupts and Registers

Type	Name	Reset† Value	Type	Name	Reset† Value
Interrupt	MIBF⟨0,1⟩	0	Register	**ILEN⟨0,1⟩**	0xFFF (−1)
	MOBE⟨0,1⟩	1		**ILIM⟨0,1⟩**	—
Register	**miwp⟨0,1⟩**	0x000		**OLIM⟨0,1⟩**	
	morp⟨0,1⟩	0x000		**OBAS⟨0,1⟩**	
	OLEN⟨0,1⟩	0x000		**IBAS⟨0,1⟩**	

† Either pin reset or execution of an MIOU⟨0,1⟩ RESET command.

Hardware Architecture (continued)

Modular I/O Units (MIOUs) (continued)

I/O Buffer Configuration

The application allocates a portion of IORAM$\langle 0,1 \rangle$ for an input (output) buffer by programming the input (output) base register **IBAS$\langle 0,1 \rangle$** (**OBAS$\langle 0,1 \rangle$**) and the input (output) limit register **ILIM$\langle 0,1 \rangle$** (**OLIM$\langle 0,1 \rangle$**). The base register specifies the first IORAM$\langle 0,1 \rangle$ location in the buffer and the limit register specifies the last IORAM$\langle 0,1 \rangle$ location in the buffer. The size of the input buffer is **ILIM$\langle 0,1 \rangle$** – **IBAS$\langle 0,1 \rangle$** + 1. The size of the output buffer is **OLIM$\langle 0,1 \rangle$** – **OBAS$\langle 0,1 \rangle$** + 1. MIOU$\langle 0,1 \rangle$ circularly advances **miwp$\langle 0,1 \rangle$** (**morp$\langle 0,1 \rangle$**) within the frame defined by the input (output) base and input (output) limit registers. Figure 17 illustrates the input and output buffer configuration.
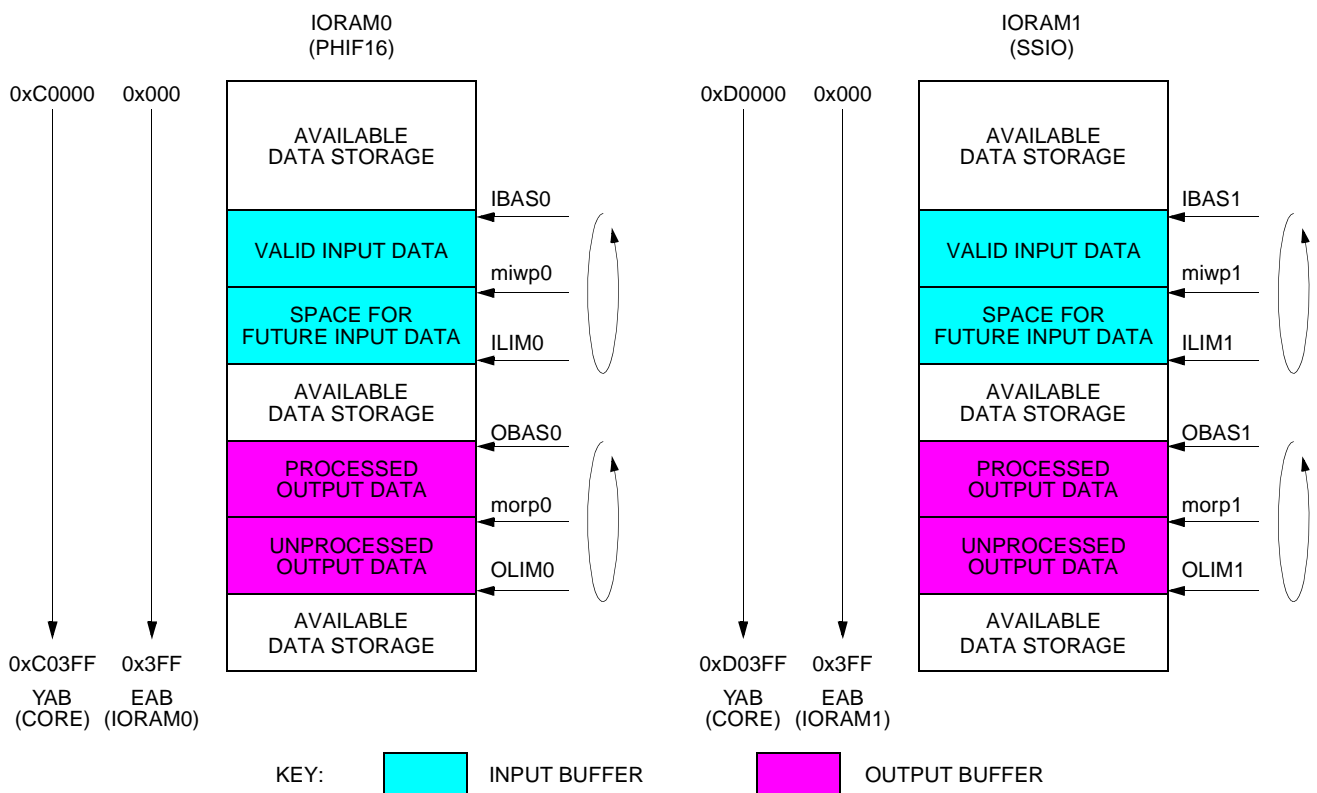
Figure 17. Input and Output Buffer Configuration in IORAM$\langle 0,1 \rangle$

The following example code segment initializes the **IBAS0** register:

```
#define ibase0 0x0100        /* IORAM0 location 0x100 (parameter)       */
#define WRibase0 0x0000      /* MIOU command to load IBAS0 (opcode)     */
a3 = WRibase0 | ibase0       /* OR to concatenate opcode and parameter  */
mcmd0 = a3                   /* Issue command IBAS0_LD                   */
```

MIOU$\langle 0,1 \rangle$ increments **miwp$\langle 0,1 \rangle$** each time it transfers an input sample from the peripheral to IORAM$\langle 0,1 \rangle$. When **miwp$\langle 0,1 \rangle$** equals **ILIM$\langle 0,1 \rangle$**, MIOU$\langle 0,1 \rangle$ loads **miwp$\langle 0,1 \rangle$** with the contents of **IBAS$\langle 0,1 \rangle$** at the completion of the following input transaction. MIOU$\langle 0,1 \rangle$ increments **morp$\langle 0,1 \rangle$** each time it transfers an output sample from IORAM$\langle 0,1 \rangle$. When **morp$\langle 0,1 \rangle$** equals **OLIM$\langle 0,1 \rangle$**, MIOU$\langle 0,1 \rangle$ loads **morp$\langle 0,1 \rangle$** with the contents of **OBAS$\langle 0,1 \rangle$** at the completion of the following output transaction.

Lucent Technologies Inc. 45

Hardware Architecture (continued)

Modular I/O Units (MIOUs) (continued)

Length Counters and MIOU Interrupts

ILEN$\langle$0,1$\rangle$ is the input length counter register that contains a 12-bit two's complement number. It contains an initial value of –1 following reset or execution of a RESET$\langle$0,1$\rangle$ command. Execution of an ILEN$\langle$0,1$\rangle$_UP command adds the command's parameter value to **ILEN$\langle$0,1$\rangle$** and causes MIOU$\langle$0,1$\rangle$ to begin input processing. MIOU$\langle$0,1$\rangle$ decrements **ILEN$\langle$0,1$\rangle$** each time it transfers an input sample from the peripheral to the IORAM$\langle$0,1$\rangle$. The input buffer full interrupt MIBF$\langle$0,1$\rangle$ is asserted when **ILEN$\langle$0,1$\rangle$** makes a transition from 0 to –1. This provides a means of input flow control (see DMA Input Flow Control for more information). After MIBF$\langle$0,1$\rangle$ is asserted, MIOU$\langle$0,1$\rangle$ continues input processing and continues to decrement **ILEN$\langle$0,1$\rangle$** so that core and MIOU$\langle$0,1$\rangle$ processing is concurrent. The software must ensure that the content of **ILEN$\langle$0,1$\rangle$** is within the range +1024 to –1023. If **ILEN$\langle$0,1$\rangle$** exceeds this range, MIBF$\langle$0,1$\rangle$ is not valid, the MIOU$\langle$0,1$\rangle$ operation is undefined, and the software must execute a RESET$\langle$0,1$\rangle$ command.

OLEN$\langle$0,1$\rangle$ is the output length counter register that contains an 11-bit unsigned number. It contains an initial value of 0 following reset or execution of a RESET$\langle$0,1$\rangle$ command. Execution of an OLEN$\langle$0,1$\rangle$_UP command adds the command's parameter value to **OLEN$\langle$0,1$\rangle$**. If an initial ILEN$\langle$0,1$\rangle$_UP command[1] has been previously executed, the execution of OLEN$\langle$0,1$\rangle$_UP causes MIOU$\langle$0,1$\rangle$ to begin output processing. If an initial ILEN$\langle$0,1$\rangle$_UP command has not been previously executed, then MIOU$\langle$0,1$\rangle$ does not begin output processing until an ILEN$\langle$0,1$\rangle$_UP command is issued. MIOU$\langle$0,1$\rangle$ decrements **OLEN$\langle$0,1$\rangle$** each time it transfers an output sample from the IORAM$\langle$0,1$\rangle$ to the peripheral. The output buffer empty interrupt MOBE$\langle$0,1$\rangle$ is asserted and MIOU$\langle$0,1$\rangle$ stops output processing when **OLEN$\langle$0,1$\rangle$** reaches 0. This provides a means of output flow control (see DMA Output Flow Control for more information). The software must ensure that the contents of **OLEN$\langle$0,1$\rangle$** does not exceed 1024. If **OLEN$\langle$0,1$\rangle$** exceeds 1024, the MIOU$\langle$0,1$\rangle$ operation is undefined and the software must execute a RESET$\langle$0,1$\rangle$ command.

Table 20 summarizes the MIOU interrupts MIBF$\langle$0,1$\rangle$ and MOBE$\langle$0,1$\rangle$.

Table 20. MIOU Interrupts

Interrupt	Condition to Assert	Condition to Clear
MIBF$\langle$0,1$\rangle$	**ILEN$\langle$0,1$\rangle$** decrements below zero.	Software issues ILEN$\langle$0,1$\rangle$_UP command resulting in **ILEN$\langle$0,1$\rangle$** $\geq$ 0.
		Software issues a RESET$\langle$0,1$\rangle$ command. Pin is reset.
MOBE$\langle$0,1$\rangle$	**OLEN$\langle$0,1$\rangle$** decrements to zero.	Software issues OLEN$\langle$0,1$\rangle$_UP command resulting in **OLEN$\langle$0,1$\rangle$** > 0.
	Software issues a RESET$\langle$0,1$\rangle$ command.	
	Pin reset.	

DMA Input Flow Control

Prior to configuring the MIOU input control registers (**miwp$\langle$0,1$\rangle$**, IBAS$\langle$0,1$\rangle$, and ILIM$\langle$0,1$\rangle$), the user's software must execute the RESET$\langle$0,1$\rangle$ command. This ensures that MIOU$\langle$0,1$\rangle$ peripheral service operations do not disturb the register configuration. The software then executes an ILEN$\langle$0,1$\rangle$_UP command to begin input operations.

The core and MIOU$\langle$0,1$\rangle$ cooperate to manage the input flow by updating **ILEN$\langle$0,1$\rangle$**. Typically, software initializes **ILEN$\langle$0,1$\rangle$** with the logical buffer size (number of samples), L1, of the first input transaction. When MIBF$\langle$0,1$\rangle$ is asserted, software processes the first logical buffer (using L1) and issues an ILEN$\langle$0,1$\rangle$_UP command with a parameter equal to the number of samples in the next logical buffer (L2). MIOU$\langle$0,1$\rangle$ and core processing are concurrent, so the MIOU$\langle$0,1$\rangle$ fills the new buffer while the first buffer is processed by the core.

---

1. The initial ILEN$\langle$0,1$\rangle$_UP command after reset activates MIOU$\langle$0,1$\rangle$ and its attached peripheral.

Hardware Architecture (continued)

Modular I/O Units (MIOUs) (continued)

DMA Input Flow Control (continued)

The ILEN$\langle 0,1 \rangle$_UP command is an accumulating operation that permits I/O and core processing to be overlapped and the logical buffer structure to be enforced by synchronizing MIBF$\langle 0,1 \rangle$ interrupts. If ILEN$\langle 0,1 \rangle$_UP operations (L1 and L2) are issued without synchronizing with an intervening MIBF$\langle 0,1 \rangle$, the subsequent MIBF$\langle 0,1 \rangle$ interrupt occurs when the (L1 plus L2) samples are processed.

The assertion of MIBF$\langle 0,1 \rangle$ does not necessarily imply that all input buffer resources are exhausted (as IBF does for the SSIO). MIBF$\langle 0,1 \rangle$ is a flow control signal and does not affect MIOU$\langle 0,1 \rangle$ processing of input or output data.

DMA Output Flow Control

The core and MIOU$\langle 0,1 \rangle$ cooperate to manage the output flow by updating **OLEN$\langle 0,1 \rangle$**. Typically, software initializes **OLEN$\langle 0,1 \rangle$** with the logical buffer size (number of samples), L1, of the first input transaction. When MOBE$\langle 0,1 \rangle$ is asserted, software processes the first logical buffer (using L1) and issues an OLEN$\langle 0,1 \rangle$_UP command with a parameter equal to the number of samples in the next logical buffer (L2). MIOU$\langle 0,1 \rangle$ and core processing are concurrent, so the MIOU$\langle 0,1 \rangle$ fills the new buffer while the first buffer is processed by the core.

MIOU$\langle 0,1 \rangle$ produces a busy flag MBUSY$\langle 0,1 \rangle$ that indicates that it has unfinished output operations pending. When this signal is cleared, all scheduled output transfers are complete and the core can safely enter low-power standby mode. MIOU0 produces the software-visible MBUSY0 condition flag in **alf** register bit 4. MIOU1 produces the software-visible MBUSY1 condition flag in **alf** register bit 5. (See Table 37 on page 85 and Table 42 on page 91.)

MIOU Performance

The MIOU supports a maximum throughput of a single 16-bit input word or a single 16-bit output word every four DSP clock periods (maximum sustained throughput of CLK/4 words/second).

External timing constraints may not permit an external device to drive at these rates. In addition, this maximum rate is reduced by core-MIOU IORAM collisions.

Powering Down an MIOU

An MIOU remains powered up and operational in low-power standby mode. (Its clock remains running and is not stopped when AWAIT (**alf**[15]) is set.)

The program powers down an MIOU by setting MIOU0 (bit 2) or MIOU1 (bit 3) of the **powerc** register (see Table 65 on page 106). If an MIOU is powered down, then some of its internal state information is lost. Therefore, an MIOU should be powered down only under one of the following two conditions:

1. The MIOU is not required in the application.
2. After powering down the MIOU and then powering it up, the application reinitializes the MIOU by executing an MIOU RESET command (see MIOU Commands beginning on page 43).

Hardware Architecture (continued)

Modular I/O Units (MIOUs) (continued)

MIOU Command Latencies

As a consequence of the pipelined IDB (internal data bus), there is a write-to-read latency for data move instructions that access peripheral (off-core) registers. DSP initiated MIOU operations incur a delay before completion of the operation can be observed in a DSP flag or register. These latencies are summarized in Table 21.

Table 21. MIOU Command Latencies

MIOU Command†	Subsequent Instruction†	Latency (Cycles)	Example	

mcmd⟨0,1⟩ = ⟨ILEN_UP, OLEN_UP, RESET⟩	**ireturn**
	
	(return from interrupt service routine)
		4	`mcmd0=0x4010`
			`4*nop`
			`ireturn`
				ILEN_UP command clears MIBF0 request. 4 **nop**s are needed to avoid unintentional re-entry into ISR.

	ins = ⟨REG, MEM⟩
	
	(clear interrupt pending bit within a polling routine)
		6	`mcmd1=0x6000`
			`6*nop`
			`ins=0x00008`
			`a0=ins`
				RESET command clears MIBF1 request and sets MOBE1 request. 6 **nop**s are needed before MIBF1 bit in **ins** can be cleared.

mcmd⟨0,1⟩ = OLEN_UP	**⟨REG, MEM⟩ = alf**
	
	(poll MBUSY1 in **alf**)
		5	`mcmd1=0x5001`
			`5*nop`
			`if mbusy1 goto wait`
				Five instruction cycles are required between an **OLEN** update and the test of the MBUSY1 flag for completion of the corresponding output operation.

	if mbusy⟨0,1⟩ INSTR (poll MBUSY1 with conditional instruction.)			

† Key to these columns: REG is any register, MEM is any memory location, ILEN_UP is a value (immediate, register contents, or memory location contents) such that bits [15:12] are 0x4, OLEN_UP is a value (immediate, register contents, or memory location contents) such that bits [15:12] are 0x5, and INSTR is any conditional instruction.

Hardware Architecture (continued)

Simple Serial I/O (SSIO) Unit

The SSIO provides a 26 Mbits/s serial interface to many codecs and signal processors with few, if any, additional components. The high-speed, double-buffered port supports back-to-back transmissions of data. The SSIO is configurable as active or passive and is a DMA peripheral that interfaces to IORAM1 through MIOU1.

There are four active clock speeds selectable by ACLK (bits 7 and 8) of the simple serial I/O control register (**SSIOC**). (See Table 70 on page 110.)

A bit-reversal mode under control of **SSIOC** register bit 6 provides compatibility with either the most significant bit (MSB) first or least significant bit (LSB) first serial I/O formats.

The serial data can be internally looped back (DO looped back to DI) by setting the SSIO loopback control bit, SIOLB (bit 9) of the **ioc** register. SIOLB affects only the SSIO.

Setting data out delay (DODLY), bit 10, of **SSIOC** to 1 delays DO by one phase of OCK so that DO changes on the falling edge of OCK instead of the rising edge (DODLY = 0). This reduces the time available for DO to drive DI and to be valid for the rising edge of ICK, but increases the hold time on DO by half a cycle of OCK.

A falling edge on the SYNC input pin causes the resynchronization of the active input load (ILD) and output load (OLD) generators. This input has typically 0.7 V hysteresis. If SYNC is not used, it must be tied low.

Programmable Modes

SSIOC controls the programmable modes of operation for the SSIO. This register, shown in Table 70 on page 110, is used to set the port into various configurations. Both input and output operations can be independently configured as either active or passive. When active, the DSP16210 generates load and clock signals. When passive, load and clock signal pins are inputs.

Since input and output can be independently configured, the SSIO has four different modes of operation.

The **SSIOC** register is also used to select the frequency of active clocks for the SSIO. Finally, **SSIOC** is used to configure the serial I/O data formats. The data can be 8 or 16 bits long, and can also be input/output MSB or LSB first. Input and output data formats can be independently configured.

The **SSIOC** register is programmed through MIOU1.

Parallel Host Interface (PHIF16)

The DSP16210 has a 16-bit parallel host bus interface for rapid transfer of data with external devices. PHIF16 is a DMA peripheral that interfaces to IORAM0 through MIOU0.

This parallel port is passive (data strobes provided by an external device) and supports either *Motorola* or *Intel* microcontroller protocols. The PHIF16 can be configured by software to operate with either an 8-bit or 16-bit external interface. (See the **PHIFC** register, Table 63 on page 104.)

In 8-bit external configuration, PHIF16 provides for 8-bit or 16-bit logical data transfers. 8-bit data is right-justified. As a flexible host interface, it requires little or no glue logic to interface to other devices (e.g., microcontrollers, microprocessors, or another DSP).

The logical data path of the PHIF16 consists of a 16-bit input register, **PDX**(in), and a 16-bit output register, **PDX**(out). **PDX**(in) is loaded with host data from the 16-bit data bus PB[15:0]. **PDX**(out) is loaded by MIOU0 with output data from the IORAM0 location addressed by the MIOU output read pointer 0 (**morp0**) register.

Two output pins, parallel input buffer full (PIBF) and parallel output buffer empty (POBE), indicate the state of the **PDX** buffers. In addition, there are two registers used to control and monitor the PHIF's operation: the parallel host interface control register (**PHIFC**, see Table 63 on page 104), and the PHIF16 status register (**PSTAT**, see Table 24 on page 51). The **PSTAT** register, which reflects the state of the PIBF and POBE flags, can only be read by an external device when the PSTAT input pin is asserted. The **PHIFC** register defines the programmable options for this port and is programmed through MIOU0 using PCTL_LD, the peripheral control load command (see Table 18 on page 44).

Hardware Architecture (continued)

Parallel Host Interface (PHIF16) (continued)

The function of the pins PIDS and PODS is programmable to support both the *Intel* and *Motorola* protocols. The PCSN pin is an input that, when low, acts as a chip-select to enable PIDS and PODS (or PRWN and PDS, depending on the protocol used). If PCSN is low, the assertion of PIDS and PODS by an external device causes the PHIF16 to recognize a host request. If MIOU0 has been properly programmed, it responds to the host request by either filling **PDX**(out) or emptying **PDX**(in). While PCSN is high, the DSP16210 ignores any activity on PIDS and/or PODS. If a DSP16210 is intended to be continuously accessed through the PHIF16 port, PCSN should be grounded.

Programmability

The PHIF16 external interface is configured for 8-bit or 16-bit external operation using bit 7 of the **PHIFC** register (PCFIG).

In the 16-bit external configuration, every completion of an input (host) or output (MIOU0) transaction asserts the external PIBF or POBE conditions.

In the 8-bit external configuration, the PHIF16 interface is programmed for 8-bit or 16-bit logical data transfers using bit 0, PMODE, of the **PHIFC** register. Setting PMODE selects 16-bit logical transfer mode. An input pin controlled by the host, PBSEL, determines an access of either the high or low byte. The assertion level of the PBSEL input pin is configurable in software using bit 3 of the **PHIFC** register, PBSELF. Table 22 on page 51 summarizes the port's output functionality as controlled by the PSTAT and PBSEL pins and the

PBSELF and PMODE fields. Table 23 on page 51 summarizes the port's input functionality.

In the 8-bit external configuration and 16-bit logical mode, PHIF16 assertion of the PIBF and POBE flags is based on the status of the PBSELF bit in the **PHIFC** register.

- If PBSELF is zero, the PIBF and POBE flags are set after the high byte is transferred.
- If PBSELF is one, the flags are set after the low byte is transferred.

In the 8-bit external configuration and 8-bit logical mode, only the low byte is accessed, and every completion of an input or output access sets PIBF or POBE.

Bit 1 of the **PHIFC** register, PSTROBE, configures the port to operate either with an *Intel* protocol where only the chip select (PCSN) and either of the data strobes (PIDS or PODS) are needed to make an access, or with a *Motorola* protocol where the chip select (PCSN), a data strobe (PDS), and a read/write strobe (PRWN) are needed. PIDS and PODS are negative assertion data strobes while the assertion level of PDS is programmable through bit 2, PSTRB, of the **PHIFC** register.

Finally, the assertion level of the output pins, PIBF and POBE, is controlled through bit 4, PFLAG. When PFLAG is set low, PIBF and POBE output pins have positive assertion levels. By setting bit 5, PFLAGSEL, the logical OR of PIBF and POBE flags (positive assertion) is seen at the output pin PIBF. By setting bit 6 in **PHIFC**, PSOBEF, the polarity of the POBE flag in the status register, **PSTAT**, is changed. PSOBEF has no effect on the POBE pin.

PHIFC is programmed through MIOU0.

Hardware Architecture (continued)

Parallel Host Interface (PHIF16) (continued)

Table 22. PHIF16 Output Function

PCFIG Field	PMODE Field	PSTAT Pin	PBSEL Pin XOR PBSELF Field	PB[15:8](out)	PB[7:0](out)
0: 8-bit external	0: 8-bit logical	0	0	3-state	PDX[7:0](out)
0: 8-bit external	0: 8-bit logical	0	1	Reserved	
0: 8-bit external	0: 8-bit logical	1	0	3-state	PSTAT
0: 8-bit external	0: 8-bit logical	1	1	Reserved	
0: 8-bit external	1: 16-bit logical	0	0	3-state	PDX[7:0](out)
0: 8-bit external	1: 16-bit logical	0	1	3-state	PDX[15:8](out)
0: 8-bit external	1: 16-bit logical	1	0	3-state	PSTAT
0: 8-bit external	1: 16-bit logical	1	1	3-state	PSTAT
1: 16-bit external	0: Preserve H & L	0	X	PDX[15:8](out)	PDX[7:0](out)
1: 16-bit external	0: Preserve H & L	1	X	0x00	PSTAT
1: 16-bit external	1: Swap H & L	0	X	PDX[7:0](out)	PDX[15:8](out)
1: 16-bit external	1: Swap H & L	1	X	PSTAT	0x00

Table 23. PHIF16 Input Function

PCFIG Field	PMODE Field	PBSEL Pin XOR PBSELF Field	PDX[15:8](in)	PDX[7:0](in)
0: 8-bit external	0: 8-bit logical	0	No change	PB[7:0](in)
0: 8-bit external	0: 8-bit logical	1	Reserved	
0: 8-bit external	1: 16-bit logical	0	No change	PB[7:0](in)
0: 8-bit external	1: 16-bit logical	1	PB[7:0](in)	No change
1: 16-bit external	0: Preserve H & L	X	PB[15:8](in)	PB[7:0](in)
1: 16-bit external	1: Swap H & L	X	PB[7:0](in)	PB[15:8](in)

Table 24. PHIF16 Status (PSTAT) Register

7—2	1	0
Reserved	PIBF	POBE

Hardware Architecture (continued)

Bit Input/Output Unit (BIO)

The BIO controls the directions of eight bidirectional control I/O pins, IOBIT[7:0]. If a pin is configured as an output, it can be individually set, cleared, or toggled. If a pin is configured as an input, it can be read and/or tested.

The lower half of the **sbit** register (see Table 69 on page 109) contains current values (VALUE[7:0]) of the eight bidirectional pins IOBIT[7:0]. The upper half of the **sbit** register (DIREC[7:0]) controls the direction of each of the pins. A logic 1 configures the corresponding pin as an output; a logic 0 configures it as an input. The upper half of the **sbit** register is cleared upon reset.

The **cbit** register (see Table 45 on page 94) contains two 8-bit fields, MODE/MASK[7:0] and DATA/PAT[7:0]. The meaning of a bit in either field depends on whether it has been configured as an input or an output in **sbit**. If a pin has been configured to be an output, the meanings are MODE and DATA. For an input, the meanings are MASK and PAT(tern). Table 25 shows the functionality of the MODE/MASK and DATA/PAT bits based on the direction selected for the associated IOBIT pin.

Those pins that have been configured as inputs can be individually tested for 1 or 0. For those inputs that are being tested, there are four flags produced: ALLT (all true), ALLF (all false), SOMET (some true), and SOMEF (some false). Table 26 summarizes these flags, which can be used for conditional instructions (see Table 37 on page 85). The state of these flags can be tested, saved, or restored by reading or writing bits 0 to 3 of the **alf** register (see Table 42 on page 91).

In input mode, the IOBIT[7:0] inputs are synchronized to the internal DSP clock (CLK) before the flags are generated or the input data is transferred to the core through the **sbit** register. In output mode, the flags are updated each time the **cbit** register is written.

Table 25. BIO Operations

DIREC[n][†]	MODE/ MASK[n][†]	DATA/ PAT[n][†]	Action
1 (Output)	0	0	Clear
1 (Output)	0	1	Set
1 (Output)	1	0	No Change
1 (Output)	1	1	Toggle
0 (Input)	0	0	No Test
0 (Input)	0	1	No Test
0 (Input)	1	0	Test for Zero
0 (Input)	1	1	Test for One

[†] $0 \leq n \leq 7$.

If a BIO pin is switched from being configured as an output to being configured as an input and then back to being configured as an output, the pin retains the previous output value. After writing **sbit** to change a pin from an output to an input, one instruction cycle of latency is required before the **sbit** VALUE field is updated. If a pin is configured as an output and **cbit** is written to change the output value, two cycles of latency are required before the **sbit** VALUE field is updated to reflect the change to **cbit**.

Table 26. BIO Flags

Condition		SOMEF (alf[3])	SOMET (alf[2])	ALLF (alf[1])	ALLT (alf[0])
All or some of the IOBIT[7:0] pins are configured as inputs[†].	All tested inputs match the pattern[‡].	0	1	0	1
	No tested inputs match the pattern[§].	1	0	1	0
	Some (but not all) of the tested inputs match the pattern[††].	1	1	0	0
	No inputs are tested[‡‡].	0	0	1	1
All IOBIT[7:0] pins are configured as outputs[§§].		0	0	1	1

[†] For at least one pin IOBIT[n], DIREC[n] = 0.
[‡] For every pin, IOBIT[n] with DIREC[n] = 0 and MASK[n] = 1, IOBIT[n] = PAT[n].
[§] For every pin, IOBIT[n] with DIREC[n] = 0 and MASK[n] = 1, IOBIT[n] ≠ PAT[n].
[††] For at least one pin, IOBIT[n] with DIREC[n] = 0 and MASK[n] = 1, IOBIT[n] = PAT[n], and for at least one pin IOBIT[n] with DIREC[n] = 0 and MASK[n] = 1, IOBIT[n] ≠ PAT[n].
[‡‡] For all pins, IOBIT[n] with DIREC[n] = 0, MASK[n] is 0.
[§§] Bits DIREC[7:0] are all ones.

Hardware Architecture (continued)

Bit Input/Output Unit (BIO) (continued)

Two instruction cycles of latency are required following a BIO **cbit** register write operation before the new BIO flags are available:

```
cbit= 0x0302
2*nop /* nops or other instructions */
if allt goto OK/* New Flags are visible*/
```

Pin Multiplexing

Four of the eight BIO signals (IOBIT[7:4]) are multiplexed with the four vectored interrupt ID signals (VEC[3:0]) onto four package pins. Upon reset, VEC[3:0] are connected to the pins while IOBIT[7:4] are disconnected. Setting bit 8, EBIO, of the **ioc** register connects IOBIT[7:4] to the pins and disconnects VEC[3:0]. Note that VEC0 corresponds to IOBIT7, VEC1 corresponds to IOBIT6, VEC2 corresponds to IOBIT5, and VEC3 corresponds to IOBIT4.

Timers

The DSP16210 contains two identical independent timers, TIMER0 and TIMER1. TIMER$\langle 0,1 \rangle$ interrupts the core after a programmed delay or repetitively at a programmed interval.

Each timer contains a 16-bit control register (**timer$\langle 0,1 \rangle$c**), down counter, period register, and a 4-bit prescaler. The 16-bit **timer$\langle 0,1 \rangle$** running count register corresponds to the down counter and period register. If **timer$\langle 0,1 \rangle$** is read, it returns the output of the down counter. If **timer$\langle 0,1 \rangle$** is written, the write value is loaded into the down counter and the period register simultaneously. The prescaler divides the internal clock (CLK) by a programmed value in the range 2 to 65536. The down counter decrements every cycle of the prescaled clock. When it reaches zero, the timer asserts its interrupt (TIME$\langle 0,1 \rangle$). The interrupt delay is a function of the CLK frequency, the initial value programmed in **timer$\langle 0,1 \rangle$**, and the prescale value. For periodic timed interrupts, the timer can be programmed to repetitively reload the down counter with the contents of the period register.

See Table 71 and Table 72 on page 111 for descriptions of **timer$\langle 0,1 \rangle$** and **timer$\langle 0,1 \rangle$c**.

By default after reset, the timers are powered up and the down counter holds its current count. To save power if the timer is not in use, set the PWR_DWN bit

of **timer$\langle 0,1 \rangle$c**. Setting **powerc**[TIMER$\langle 0,1 \rangle$] (see Table 65 on page 106) has the same effect as setting that timer's PWR_DWN bit.

Assuming the timer is powered up, setting the COUNT bit of **timer$\langle 0, 1 \rangle$c** enables the clock to the down counter. Clearing COUNT causes the counter to hold its current value.

The PRESCALE[3:0] field of **timer$\langle 0,1 \rangle$c** selects one of 16 possible clock rates for the input clock to the down counter (see Table 72 on page 111). The clock rate is the frequency of CLK divided by 2^{N+1}, where N is PRESCALE[3:0] and ranges from 0 to 15.

To operate the timer, the software writes a value to **timer$\langle 0,1 \rangle$** and sets the COUNT bit of **timer$\langle 0,1 \rangle$c** (the remaining fields of **timer$\langle 0,1 \rangle$c** must also be programmed appropriately). This causes the down counter to start decrementing. When the counter reaches zero, a vectored interrupt to program address **vbase** + *offset*[1] is issued, providing the appropriate timer interrupt is enabled[2] and no higher priority interrupt is pending or being serviced. If the RELOAD bit of **timer$\langle 0,1 \rangle$c** is 0, the timer stops decrementing the counter when it reaches zero. Software can restart the timer by writing a nonzero value to **timer$\langle 0,1 \rangle$**. If RELOAD is 1, the timer reloads the counter from the period register and the counter resumes decrementing, resulting in repetitive periodic interrupts.

Software can start and stop the timer at any time[3] by setting and clearing the COUNT bit. Software can read and write **timer$\langle 0,1 \rangle$** at any time[3]. Due to pipeline stages, stopping and starting the timers can result in an error at one count or prescaled period.

When the DSP16210 is reset, the **timer$\langle 0,1 \rangle$c** and **timer$\langle 0,1 \rangle$** registers and counters are cleared. This powers up the timer, sets the prescale value to CLK/2, disables the clock to the down counter, and turns off the reload feature. The act of resetting the chip does not cause a timer interrupt.

Note: The timer must be powered up (PWR_DWN = 0 and **powerc**[1,0] = 0) in order to read or write the **timer$\langle 0,1 \rangle$** register. If **timer$\langle 0,1 \rangle$** is read after device reset without first being written, a value of all zeros is returned. However, the initial count value and period are not cleared on reset—to clear them, the software must write **timer$\langle 0,1 \rangle$** with all zeros.

---

1. *offset* is 0x34 for TIMER0 and 0x38 for TIMER1.
2. The programmer enables the TIMER0 interrupt by setting **inc0** bits 18 and 19 to a priority. The programmer enables the TIMER1 interrupt by setting **inc1** bits 0 and 1 to a priority. See Table 52 on page 97 for details.
3. The timer must be powered up.

Hardware Architecture (continued)

Hardware Development System (HDS)

The HDS is an on-chip hardware module available for debugging assembly-language programs that execute on the DSP16000 core at the core's rated speed. The main capability of the HDS is in allowing controlled visibility into the core's state during program execution.

The fundamental steps in a debugging process, involving the HDS, include the following:

1. Setup: Download program code and data into the correct memory regions and set breakpointing conditions.

2. Run: Start execution or single step from a desired starting point (i.e., allow device to run under simulated or real-time conditions).

3. Break: Break program execution on satisfying breakpointing conditions; upload and allow user accessibility to internal state of the device and its pins.

4. Resume: Resume execution (normally or single step) after hitting a breakpoint and finally upload internal state at the end of execution.

The powerful debugging capability of the HDS is made possible by breaking program execution on complex breakpointing conditions. A complex breakpointing condition, for example, may be an instruction that executes from a particular instruction-address location (or from a particular instruction-address range such as a subroutine) and accesses a coefficient/data element that matches a particular pattern from a memory location (or from a memory region such as inside an array or outside an array). The complex conditions can also be chained to form more complex breakpointing conditions. For example, a complex breakpointing condition can be defined as the back-to-back execution of two different subroutines.

The HDS also provides a debugging feature that allows a finite number of initial complex breakpointing conditions to optionally be ignored. The number of condi-

tions ignored is programmable by the user.

An intelligent trace mechanism for recording discontinuity points during program execution is also available in the HDS. This mechanism allows unambiguous reconstruction of program flow involving discontinuity points such as gotos, calls, returns, and interrupts. The trace mechanism compresses single-level loops and records them as a single discontinuity. This feature prevents single-level loops from filling up the trace buffers. Also, cache loops do not get registered as discontinuities in the trace buffers. Therefore, two-level loops with inner cache loops are registered as a single discontinuity.

The HDS supports single stepping through instructions without requiring the use of a watchpoint register.

A 32-bit cycle counter is provided for accurate code profiling during program development. This cycle counter can optionally be used to break program execution after a user-specified number of clock cycles.

JTAG Test Port

JTAG is an on-chip hardware module that controls the HDS. All communication between the HDS software, running on the host computer, and the on-chip HDS is in a bit-serial manner through the TAP (test access port) of the device. The TAP pins, which are the means of communicating test information into and out of the device, consist of TDI (test data input), TDO (test data output), TMS (test mode select), TCK (test clock), and TRST (TAP controller reset). The registers in the HDS are connected in different scan paths between the TDI (input port) and TDO (output port) pins of the TAP. JTAG instructions have been reserved to allow read and write operations to be performed between JTAG and the register chains of the HDS.

The set of test registers include the JTAG identification register (ID), the boundary-scan register, and the scannable peripheral registers. All of the device's inputs and outputs are incorporated in a JTAG scan path as shown in Table 27 on page 55.

Hardware Architecture (continued)

JTAG Test Port (continued)

Table 27. JTAG Boundary-Scan Register

Note: The direction of shifting is from TDI to cell 121 to cell 120 . . . to cell 0 to TDO.

Cell	Type[†]	Signal Name/Function	Control Cell	Cell	Type[†]	Signal Name/Function	Control Cell	Cell	Type[†]	Signal Name/Function	Control Cell
0	I	CKI‡	—	54	I	Reserved	—	94	I	DOEN	—
8—1	O	AB[7:0]	9	55	I	EOEB	—	95	DC	IOBIT0 direction control	—
9	OE	AB[15:0], IACK, PIBF, POBE, EOBE, EIBF, OBE, and IBF 3-state control	—	56	I	PIDS	—	96	I/O	IOBIT0	95
17—10	O	AB[15:8]	9	57	I	PCSN	—	97	DC	IOBIT1 direction control	—
18	I	EXM	—	58	I	PSTAT	—	98	I/O	IOBIT1	98
19	OE	RWN, EROM, ERAMLO, ERAMHI, ERAM, IO, and CKO 3-state control	—	59	I	PBSEL	—	99	DC	IOBIT2 direction control	—
20	O	RWN	19	60	I	PODS	—	100	I/O	IOBIT2	99
21	O	EROM	19	61	O	PIBF	9	101	DC	IOBIT3 direction control	—
22	O	ERAMLO	19	62	O	POBE	9	102	I/O	IOBIT3	101
23	O	ERAMHI	19	63	DC	PB[7:0] direction control	—	103	DC	VEC3/IOBIT4 direction control	—
24	O	ERAM	19	71—64	I/O	PB[7:0]	63	104	I/O	VEC3/IOBIT4	103
25	O	IO	19	72	DC	PB[15:8] direction control	—	105	DC	VEC2/IOBIT5 direction control	—
33—26	I/O	DB[7:0]	34	80—73	I/O	PB[15:8]	72	106	I/O	VEC2/IOBIT5	105
34	DC	DB[15:0] direction control	—	81	O	OBE	9	107	DC	VEC1/IOBIT6 direction control	—
42—35	I/O	DB[15:8]	34	82	O	IBF	9	108	I/O	VEC1/IOBIT6	107
43	O	EOBE	9	83	I	DI	—	109	DC	VEC0/IOBIT7 direction control	—
44	O	EIBF	9	84	DC	ILD direction control	—	110	I/O	VEC0/IOBIT7	109
45	I	EDI	—	85	I/O	ILD	84	111	I	READY	—
46	I	EIFS	—	86	DC	ICK direction control	—	112	I	STOP	—
47	I	EIBC	—	87	I/O	ICK	86	113	I	RSTB	—
48	I	EOBC	—	88	DC	OCK direction control	—	114	O	CKO	19
49	I	EOFS	—	89	I/O	OCK	88	115	DC	TRAP direction control	—
50	I	Reserved	52	90	DC	OLD direction control	—	116	I/O	TRAP	115
51	OE	EDO 3-state control	—	91	I/O	OLD	90	117	O	IACK	9
52	O	EDO	—	92	OE	DO 3-state control	—	121—118	I	INT[3:0]	—
53	I	SYNC	—	93	O	DO	92				

† Key to this column: I = input; OE = 3-state control cell; O = output; DC = bidirectional control cell; I/O = input/output.
‡ When read with the JTAG SAMPLE instruction, CKI returns a logic one regardless of the state of the pin.

Hardware Architecture (continued)

Clock Synthesis

The DSP16210 provides an on-chip programmable clock synthesizer that can be driven by an external clock at a fraction of the desired instruction rate. Figure 18 is the synthesizer block diagram, which is based on a phase-lock loop (PLL). The terms clock synthesizer and PLL are used interchangeably.
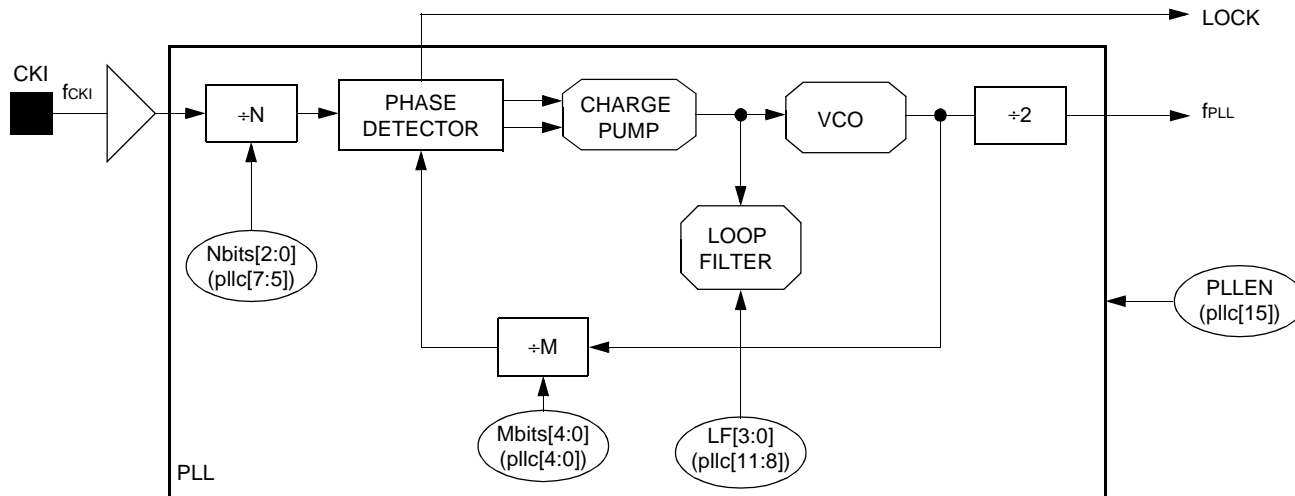
Notes:
If PLLEN is set, the PLL is enabled (powered up). If PLLEN is cleared, the PLL is disabled (powered down).

The PLL sets the LOCK flag when its output is stable. The LOCK flag is an input to CORE0 and to CORE1.

Figure 18. Clock Synthesizer (PLL) Block Diagram

Figure 19 on page 57 illustrates the internal clock selection and disable logic. The clock selection logic selects the internal clock (CLK) from one of the following three clock sources:

■ CKI: This pin is driven by an external oscillator or the pin's associated boundary-scan logic under JTAG control. If CKI is selected as the clock source, then CLK has the frequency and duty cycle of CKI.

■ PLL: The PLL generates a clock source with a programmable frequency (an M/2N multiple of the CKI clock). The PLL's output is f_{PLL}. If the PLL is selected as the clock source, then CLK has the frequency and duty cycle of the PLL output f_{PLL}.

■ Ring Oscillator: The internal ring oscillator produces a slow clock that requires no external stimulus. When the slow clock is selected as the clock source, then CLK has the frequency and duty cycle of the ring oscillator output. The core consumes less power when clocked with the slow

clock. See Table 91 on page 141 for timing characteristics of the ring oscillator.

After device reset, CKI is selected as the default clock source for the DSP16210. Setting the appropriate bits in the **pllc** and **powerc** control registers (Table 64 on page 105 and Table 65 on page 106) enables either the PLL or the ring oscillator to become the clock source. Table 28 defines the selection of the three clock sources as a function of the PLLSEL field (bit 14 of **pllc**) and the SLOWCLK field (bit 10 of **powerc**).

Table 28. Clock Source Selection

PLLSEL (pllc[14])	SLOWCLK (powerc[10])	f_{CLK}	Description
0	0	f_{CKI}	CKI pin
0	1	$f_{SLOW\ CLOCK}$	Ring Oscillator
1	X	f_{PLL}	PLL

The clock disable logic provides several methods for shutting off the internal clock to save power. See Power Management beginning on page 61 for details.

Hardware Architecture (continued)

Clock Synthesis (continued)
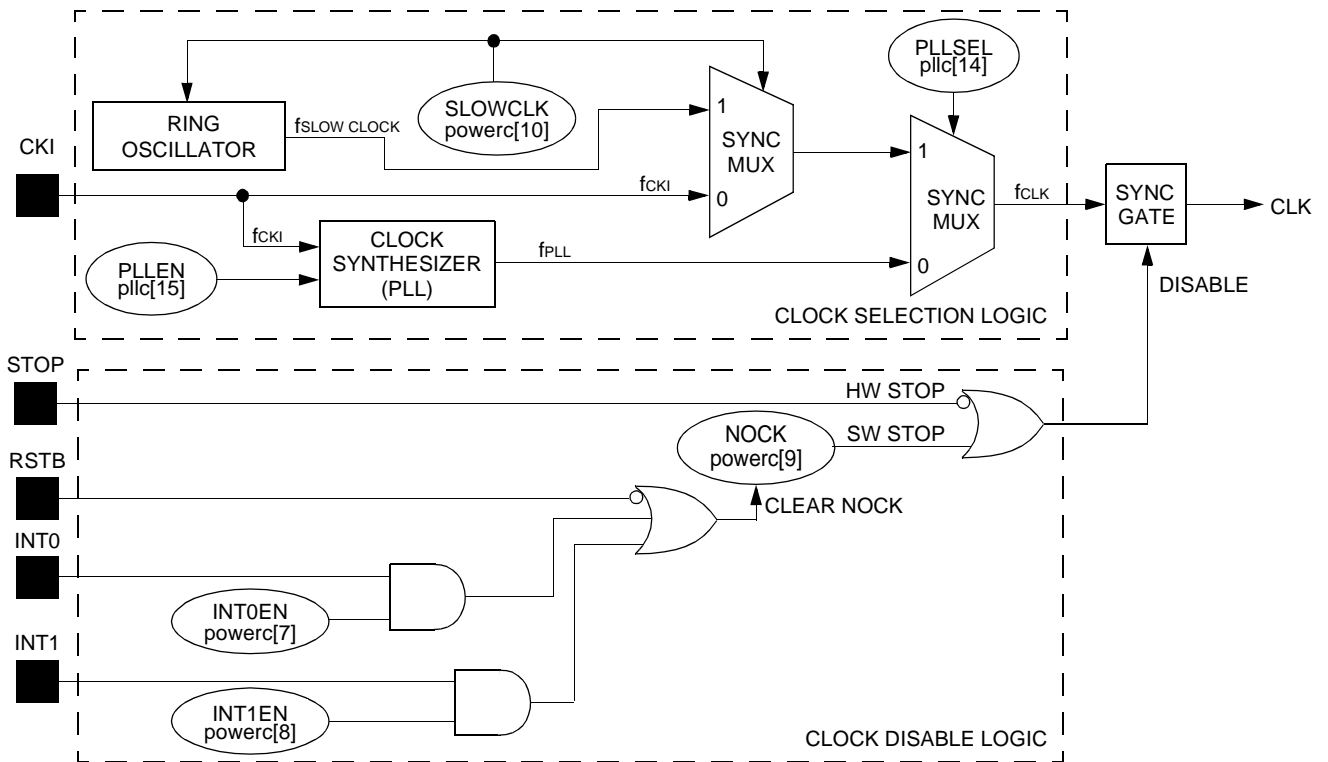
Note: The switching of the multiplexers and the synchronous gate is designed so that no partial clocks or glitching occurs.

Figure 19. Internal Clock Selection and Disable Logic

Hardware Architecture (continued)

Clock Synthesis (continued)

The clock switch module (the SYNC MUX blocks shown in Figure 19 on page 57) selects the clock source synchronously for glitch-free operation. Potential clock sources are first synchronized to the current CLK before being prioritized and acted upon by the clock switch module.

Phase-Lock Loop (PLL) Operation

Because **pllc** is cleared on reset, the PLL is initially deselected and powered down. For the PLL to operate, the following is required:

1. A clock must be applied to the CKI input pin, the input to the PLL.

2. The program must enable (power up) the PLL by setting the PLLEN bit (**pllc** bit 15). (Clearing PLLEN disables (powers down) the PLL.) The program must not select the PLL, i.e., must not set the PLLSEL bit (**pllc** bit 14), until the LOCK flag is set as described later in this section. The programming of the remaining bits of **pllc** and the frequency of CKI determine the frequency of the PLL output.

Phase-Lock Loop (PLL) Operating Frequency

The frequency of the PLL output clock (f_{PLL}) is determined by the values loaded into the 3-bit N divider and the 5-bit M divider as follows:

$$f_{PLL} = f_{CKI} \times M/2N$$

where $2 \leq M \leq 24$ and $1 \leq N \leq 8$. The maximum allowable M/N ratio is 12. If the PLL is selected as the clock source, the frequency of the internal clock (CLK) is:

$$f_{CLK} = f_{PLL} = f_{CKI} \times M/2N$$

The following requirements apply to the f_{PLL}:

- $f_{PLL} \geq f_{CKI}$.
- f_{PLL} must fall within the range defined in Table 87 on page 139. (f_{CLK} must not exceed the maximum instruction rate defined in Table 83 on page 133).

After choosing f_{PLL} and f_{CKI}, choose the lowest value for N and the appropriate value of M to obtain the desired frequency. Program M and N into the Mbits[4:0] and Nbits[2:0] fields (**pllc**[4:0] and **pllc**[7:5]) as follows:

$$Mbits[4:0] = M - 2$$

```
if (N==1)
        Nbits[2:0] = 0x7
else
        Nbits[2:0] = N - 2
```

The results of these formulas are summarized in Table 29:

Table 29. pllc Field Values Nbits[2:0] and Mbits[2:0]

N	Nbits[2:0]		M	Mbits[4:0]	
1	7	111	2	0	00000
2	0	000	3	1	00001
3	1	001	4	2	00010
4	2	010	5	3	00011
5	3	011	6	4	00100
6	4	100	•	•	•
7	5	101	•	•	•
8	6	110	24	22	10110

Program the loop filter field LF[3:0] (**pllc**[11:8]) according to Table 88 on page 139.

Phase-Lock Loop (PLL) Locking

Before selecting the PLL as the clock source, the program must ensure that the PLL has stabilized and locked to the programmed frequency. The DSP16210 indicates that the PLL has locked by setting the LOCK flag (see Table 37 on page 85 and **alf** register bit 6 in Table 42 on page 91). Once the program has checked that the LOCK flag is set, it can then safely set PLLSEL (**pllc** bit 14) to switch sources from f_{CKI} to f_{PLL} without glitching. If LOCK is cleared, the PLL output is unstable. Every time the program writes **pllc**, the LOCK flag is cleared. The LOCK flag status is tested by conditional instructions that have the qualifier **if lock**, e.g., **if lock goto pll_select**. The typical lock-in time is specified in Table 88 on page 139.

Before removing the clock from the clock input pin (CKI), the program must first deselect and power down the PLL (PLLSEL = 0 and PLLEN = 0). Otherwise, the LOCK flag is not cleared, and when the input clock is reapplied it cannot be determined when the PLL has stabilized.

Hardware Architecture (continued)

Clock Synthesis (continued)

Phase-Lock Loop (PLL) Programming Restrictions
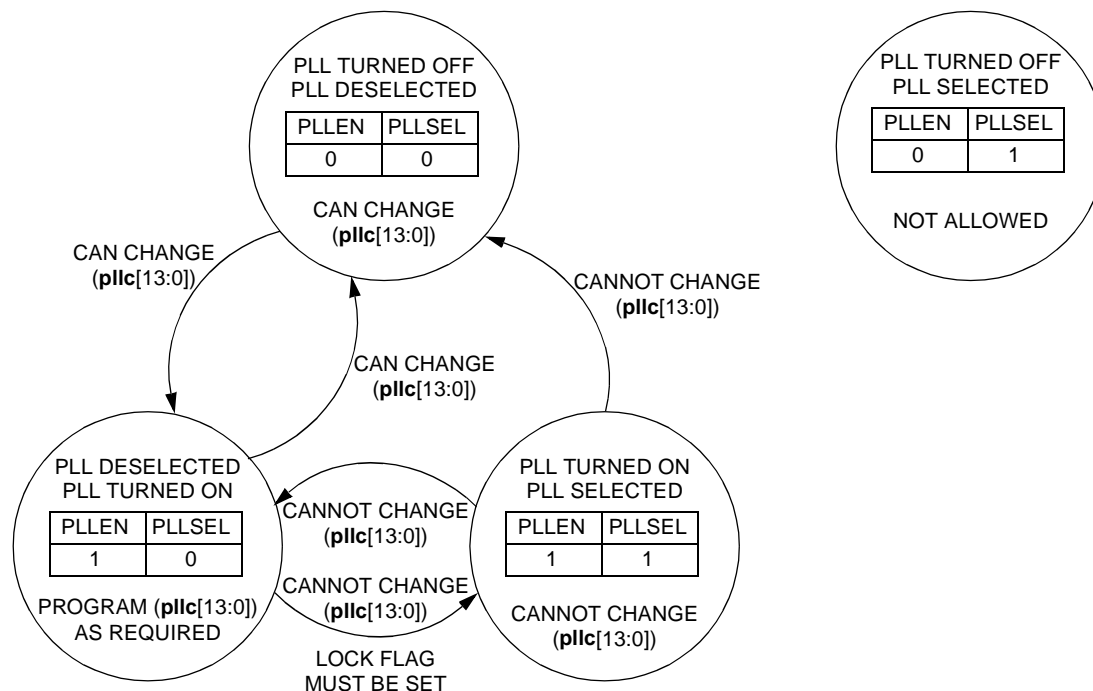
Figure 20. Allowable States and State Changes of pllc Register Fields

There are restrictions on the allowable states of the PLLEN and PLLSEL fields (**pllc**[15:14]), and on the allowable changes to these fields and the remaining fields of **pllc**. Figure 20 illustrates these restrictions, summarized below:

- Do not select the PLL if it is not enabled (PLLEN = 0 and PLLSEL = 1 is not allowed).

- Do not enable and select the PLL in one step (do not change both PLLEN from 0 to 1 and PLLSEL from 0 to 1 within a single instruction write to **pllc**). Instead, perform the following steps:

  1. Enable the PLL without selecting it, i.e., write **pllc** such that PLLEN = 1, PLLSEL = 0, and **pllc**[13:0] (Mbits[4:0], Nbits[2:0], etc.) are programmed appropriately.

  2. Wait until the LOCK flag is set.

  3. Select the PLL as the clock source, i.e., write **pllc** such that PLLEN = 1, PLLSEL = 1, and **pllc**[13:0] are programmed to the same values as in step 1.

- Do not change **pllc**[13:0] (Mbits[4:0], Nbits[2:0], etc.) while the PLL is selected (PLLSEL = 1) or while deselecting the PLL (writing **pllc** such that PLLSEL

changes from 1 to 0). To change **pllc**[13:0] if the PLL is selected:

  1. Deselect the PLL, keep it enabled, and don't change **pllc**[13:0], i.e., write **pllc** such that PLLEN = 1, PLLSEL = 0, and **pllc**[13:0] are at their old values.

  2. Program **pllc**[13:0] to the new values.

  3. Wait until the LOCK flag is set.

  4. Select the PLL as the clock source, i.e., write **pllc** such that PLLEN = 1, PLLSEL = 1, and **pllc**[13:0] are programmed to the same values as in step 2.

- The PLL can be deselected and powered down in the same instruction, i.e., both PLLEN and PLLSEL can be cleared in a single write to **pllc**, but **pllc**[13:0] cannot be changed in that same instruction (must be written with their old values).

- As long as **pllc**[13:0] remains unchanged and the PLL remains enabled (PLLEN = 1), the programmer can deselect the reselect the PLL (change PLLSEL from 1 to 0 and back again) without checking the LOCK flag status.

Hardware Architecture (continued)

Clock Synthesis (continued)

Phase-Lock Loop (PLL) Programming Example

The example in this section assumes the CKI input clock frequency is 10 MHz and the desired internal clock frequency is 100 MHz. Table 30 illustrates the calculation of the M and N values and the corresponding Mbits[4:0] and Nbits[2:0] values to be programmed into **pllc** (see Table 64 on page 105).

Table 30. Example Calculation of M and N

CKI Input Frequency	f_{CKI}	10 MHz	
CLK Frequency	f_{CLK}	100 MHz	
PLL Frequency	f_{PLL}	100 MHz	
PLL Ratio	M/2N	10	
	M/N	20	
	M	20	Mbits[4:0] = M − 2 = 18 = 0x12
	N	1	Nbits[2:0] = 7 = 0x7

The following code segment illustrates the programming, enabling, and selecting of the PLL according to the values in Table 30, assuming the PLL is initially disabled and deselected:

```
        di                     /* Disable interrupts for PLL lock (recommended)   */
        pllc = 0xa9f2          /* Enable PLL, keep it deselected, program M, N, LF */
wait:   if lock goto locked    /* Wait until LOCK flag is set                      */
        goto wait              /* While waiting, CLK = CKI = 10 MHz                */
locked: pllc = 0xe9f2          /* Select PLL clock - no other change to pllc       */
        ei                     /* Re-enable interrupts                             */
        goto start             /* User's code, now running at 100 MHz              */
```

Examples of programming the PLL and using the various power management modes are included in Power Management beginning on page 61.

Phase-Lock Loop (PLL) Frequency Accuracy and Jitter

Although the average frequency of the PLL output has almost the same relative accuracy as the input clock, noise sources within the DSP produce jitter on the PLL clock. The PLL is guaranteed to have sufficiently low jitter to operate the DSP. However, if the PLL clock is driven off the device onto the CKO pin, do not apply this clock to jitter-sensitive devices. See Table 87 on page 139 for the input jitter requirements for the PLL.

Phase-Lock Loop (PLL) Power Connections

The PLL has its own power and ground pins, VDDA and VSSA. Because the PLL contains analog circuitry, VDDA and VSSA are sensitive to supply noise. To filter supply noise, connect a dedicated decoupling capacitor from VDDA to VSSA. Depending on the characteristics of the supply noise in the particular application, a series ferrite bead or resistor might also be needed. VSSA can be connected directly to the main ground plane. This recommendation is subject to change and can be modified for specific applications depending on the characteristics of the supply noise.

Hardware Architecture (continued)

Power Management

There are three different control mechanisms for putting the DSP16210 into low-power modes: the **powerc** control register, the STOP pin, and the AWAIT bit in the **alf** register (standby mode).

Note: If the PLL is enabled (**pllc**[15] = 1), it remains running and consumes power even if the DSP16210 is in a low-power mode. For maximum power savings, disable the PLL before entering a low-power mode.

The powerc Control Register Bits

The **powerc** register has 11 bits (5 bits are reserved) that power down various portions of the chip and select the source of the internal clock (CLK):

SLOWCLK: If the program sets the SLOWCLK bit and clears the PLLSEL bit (**pllc**[14]), an internal ring oscillator is selected as the source for CLK instead of the CKI pin or the PLL. If the SLOWCLK bit is cleared, the ring oscillator is powered down. Switching of the clocks is synchronized so that no partial or short clock pulses occur. Two **nop** instructions should follow any instruction that changes the state of SLOWCLK.

NOCK: If the program sets the NOCK bit, the DSP16210 synchronously turns off CLK (regardless of whether its source is provided by the CKI pin, the PLL, or the internal ring oscillator) and stops program execution. Two **nop** instructions should follow any instruction that sets NOCK. The NOCK bit can be cleared by asserting the INT0 or INT1 pin (if the INT0EN or INT1EN bit is set). Clearing the NOCK bit in this manner allows the stopped program to resume execution from where it left off without any loss of state. If INT0EN or INT1EN is set, it is recommended that the programmer disable the corresponding interrupt in the **inc0** register before setting NOCK to avoid an unintentional interrupt due to the subsequent assertion of the INT0 or INT1 pin. After the stopped program resumes, it should clear the corresponding INT0/INT1 interrupt by writing to the **ins** register (see Clearing Interrupts on page 23). Resetting the DSP16210 by asserting the RSTB pin also clears the NOCK bit, but the stopped program cannot resume execution.

INT0EN: This bit allows the INT0 pin to asynchronously clear the NOCK bit as described above.

INT1EN: This bit allows the INT1 pin to asynchronously clear the NOCK bit as described above.

The following control bits, if set, individually power down the peripheral units, further reducing the power consumption during low-power standby mode. Figure 21 on page 62 illustrates the effect of these bits.

ESIO: This is a powerdown signal to the ESIO unit. It disables the clock input to the unit, thus eliminating any standby power associated with the ESIO. Since the gating of the clocks can result in incomplete transactions, this option can only be used in applications where the ESIO is not used or when reset is used to re-enable the ESIO unit. Otherwise, the first transaction after re-enabling the unit could be corrupted.

SSIO: This bit powers down the SSIO in the same way ESIO powers down the ESIO unit.

MIOU1: This is a powerdown signal to the MIOU1. It disables the clock input to the unit, thus eliminating any standby power associated with the MIOU1. Since the gating of the clocks can result in incomplete transactions, this option can only be used in applications where the MIOU1 is not used, or when reset is used to re-enable the MIOU1 unit. Since MIOU1 and SSIO operate independently of each other, the MIOU1 can be powered down while SSIO remains active. Before powering down MIOU1, the program should poll the MBUSY1 flag (see Table 37 on page 85) to ensure that all output activity is complete.

PHIF16: This is a powerdown signal to the PHIF16 unit. It disables the clock input to the unit, thus eliminating any standby power associated with the PHIF16. Since the gating of the clocks can result in incomplete transactions, this option can only be used in applications where the PHIF16 is not used, or when reset is used to re-enable the PHIF16 unit.

MIOU0: This is a powerdown signal to the MIOU0. It disables the clock input to the unit, thus eliminating any standby power associated with the MIOU0. Since the gating of the clocks can result in incomplete transactions, this option can only be used in applications where the MIOU0 is not used, or when reset is used to re-enable the MIOU0 unit. Since MIOU0 and PHIF16 operate independently of each other, the MIOU0 can be powered down while PHIF16 remains active. Before powering down MIOU0, the program should poll the MBUSY0 flag (see Table 37 on page 85) to ensure that all output activity is complete.

TIMER0: This is a TIMER0 disable signal that disables the clock input to the TIMER0 unit. Its function is identical to the DISABLE0 field of the **timer0c** control register. Writing a 0 to TIMER0 in the **powerc** register field will continue TIMER0 operation.

TIMER1: This bit disables the clock input to the TIMER1 unit the same way TIMER0 disables the TIMER0 unit.

Hardware Architecture (continued)

Power Management (continued)

Figure 21 illustrates the clocks to the core, the peripherals, and the CKO pin. See also Figure 19 on page 57.
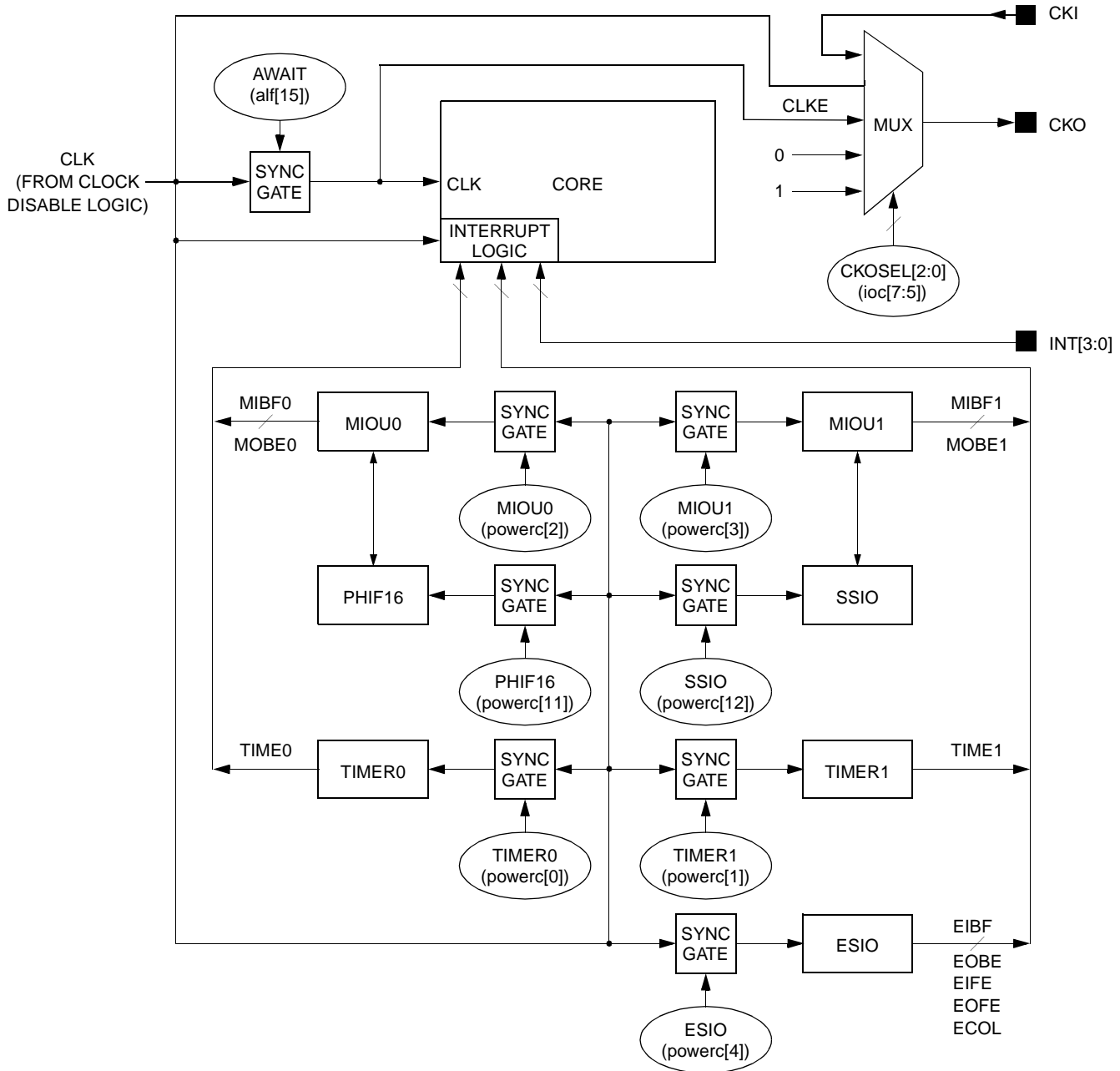
Figure 21. Power Management and Clock Distribution

Hardware Architecture (continued)

Power Management (continued)

STOP Pin

Assertion (active-low) of the STOP pin has the same effect as setting the NOCK bit in the **powerc** register. The internal clock (CLK) is synchronously disabled until STOP is returned high. Once STOP is returned high, program execution continues from where it left off without any loss of state. No device reset is required. illustrates the effect of STOP on the internal clock.

PLL Powerdown

Clearing PLLEN (bit 15 of the **pllc** register) powers down the PLL. Do not clear PLLEN if the PLL is selected as the clock source, i.e., if PLLSEL (bit 14 of

pllc) is set. See for details.

AWAIT Bit of the alf Register

Setting the AWAIT bit of the **alf** register causes the core to go into the low-power standby mode. In this mode the peripherals remain active, the PLL remains active if enabled, and the minimum core circuitry required to process an incoming interrupt remains active. Any interrupt returns the core to its previous state, and program execution continues. As long as the core is receiving a clock, whether slow or fast, it can be put into standby mode with the AWAIT bit. Once the AWAIT bit is set, the STOP pin can be used to stop and later restart the internal clock, returning to the standby state. If the internal clock is not running, however, the AWAIT bit cannot be set.

Power Management Examples

The following examples illustrate the more significant options, not an exhaustive list of options, for reducing power dissipation. The many options for reducing power include a combination of the following:

- The choice of clock source to the processor.
- Whether the user chooses to power down the peripheral units.
- Whether the internal clock is disabled through hardware or software.
- The combination of power management modes chosen.
- Whether or not the PLL or ring oscillator is enabled.

Low-Power Standby Mode with CKI Clock Input. It is assumed that the PLL is disabled (PLLEN = 0) and the processor is clocked with a high-speed clock on the CKI pin. Prior to entering low-power standby mode[1] by setting the AWAIT bit (**alf**[15]), the program reduces power by turning off all the peripherals and holding the CKO pin low.

```
         powerc=0x181f        /* Prepare for standby mode -- turn off peripherals.*/
                2*nop         /* Wait for it to take effect.                       */
                ioc=0x0040    /* Hold CKO low.                                     */
_standby:  alf=0x8000         /* Set AWAIT bit, stop internal processor clock,... */
                nop           /* interrupt circuits active.                        */
                nop           /* Needed for bedtime execution. Only standby power  */
                nop           /* consumed here until interrupt wakes up the device*/
cont:      ...                /* User code executes here                           */
         powerc=0x0000        /* Turn peripheral units back on                     */
                2*nop         /* Wait for it to take effect.                       */
                ioc=0x0000    /* CKO is free-running CLK.                          */
```

1. The program exits low-power standby mode when any enabled interrupt occurs. Therefore, it is assumed that interrupts are globally enabled and at least one interrupt is individually enabled.

Hardware Architecture (continued)

Power Management (continued)

Low-Power Standby Mode with Slow Internal Clock. It is assumed that the PLL is disabled (PLLEN = 0) and the processor is clocked with a high-speed clock on the CKI pin. Prior to entering low-power standby mode by setting the AWAIT bit (**alf**[15]), the program reduces power by turning off all the peripherals, holding the CKO pin low, and selecting the internal ring oscillator as the clock source.

Note: The ring oscillator continues to run during standby mode so there is no wake-up latency.

```
          powerc=0x1c1f    /* Prepare for standby mode--turn off peripherals,  */
          2*nop            /* select slow clock, wait for it to take effect.   */
          ioc=0x0040       /* Hold CKO low.                                    */
_standby: alf=0x8000       /* Set AWAIT bit (stop core clock; interrupt logic  */
          nop              /* active.) nops needed for bedtime execution.      */
          nop              /* Reduced standby power consumed here.             */
          nop              /* Interrupt wakes up the core.                     */
_cont:    ...              /* User code executes here.                         */
          powerc=0x0000    /* Select high-speed clock and turn on peripherals. */
          2*nop            /* Wait for it to take effect.                      */
          ioc = 0x0000     /* CKO is free-running.                             */
```

Software Stop with CKI Clock Input. It is assumed that the PLL is disabled (PLLEN = 0) and the processor is clocked with a high-speed clock on the CKI pin. Prior to performing a software stop by setting the NOCK bit (**powerc**[9]), the program reduces power by turning off all the peripherals and holding the CKO pin low. Setting the NOCK bit shuts off the internal clock and stops program execution until an interrupt on the INT0 pin restarts the internal clock. (Alternatively, INT1 or RSTB can be used to restart the clock.)

```
          powerc=0x189f    /* Prepare for software stop--set INT0EN, turn off  */
          2*nop            /* peripherals, and wait for it to take effect.     */
          di
          inc0=NO_INT0     /* Disable the INT0 interrupt (Clear inc0[11:10]).  */
          ei
          ioc = 0x0040     /* Hold CKO low.                                    */
_nock:    powerc=0x1a9f    /* Set NOCK to stop internal clock.                 */
                           /* Minimum switching power consumed here.           */
          3*nop            /* Some nops are needed.                            */
                           /* INT0 pin clears the NOCK bit; clocking resumes.  */
cont:     ...              /* User code executes here.                         */
          powerc=0x0000    /* Clear INT0EN bit and turn on peripherals.        */
          2*nop            /* Wait for it to take effect.                      */
          ins 0x0020       /* Clear the INT0 status bit.                       */
          di
          inc0=INT0        /* Safe to reenable the INT0 interrupt.             */
          ei
          ioc=0x0000       /* CKO is free-running.                             */
```

Hardware Architecture (continued)

Power Management (continued)

Low-Power Standby Mode, PLL Enabled and Selected. The PLL is enabled and selected to run at 100 MHz, assuming a constant CKI input clock of 10 MHz. Prior to entering low-power standby mode[1] by setting the AWAIT bit (**alf**[15]), the program reduces power by turning off all the peripherals and holding the CKO pin low. The PLL remains enabled and selected during standby mode and continues to dissipate power.

```
            di                 /* Globally disable interrupts for PLL lock.     */
            pllc=0xa9f2        /* pllc[15]=1 enables the PLL to run at 100 MHz   */
                               /* with CKI=10 MHz. CKI must remain running.      */
pll_buzz:                      /* Assure time for PLL to lock                    */
            if lock goto select_pll
            goto pll_buzz
select_pll:
            pllc=0xe9f2        /* pllc[14]=1 selects the PLL.                    */
            ei                 /* Globally re-enable interrupts.                 */
                  .
                  .            /* user code with CLK = 100 MHz                   */
                  .
            powerc=0x181f      /* Prepare for standby - turn off peripherals.    */
            2*nop              /* Wait for it to take effect.                    */
            ioc=0x0040         /* Hold CKO low.                                  */
_standby:   alf=0x8000         /* Set AWAIT bit (stop core clock; interrupt logic */
            nop                /* active.) nops needed for bedtime execution.    */
            nop                /* Reduced standby power plus PLL power consumed.  */
            nop                /* Interrupt wakes up the core.                   */
cont:       ...                /* User code executes here.                       */
            powerc=0x0000      /* Turn peripheral units back on.                 */
            2*nop              /* Wait for it to take effect.                    */
            ioc=0x0000         /* CKO is free-running.                           */
```

---

1. The program exits low-power standby mode when any enabled interrupt occurs. Therefore, it is assumed that interrupts are globally enabled and at least one interrupt is individually enabled.

Hardware Architecture (continued)

Power Management (continued)

Low-Power Mode Standby with Slow Internal Clock, PLL Enabled and Not Selected. The PLL is enabled to run at 100 MHz, assuming a constant CKI input clock of 10 MHz. Prior to entering low-power standby mode[1] by setting the AWAIT bit (**alf**[15]), the program reduces power by turning off all the peripherals, holding the CKO pin low, and selecting the slow internal clock as the clock source. The PLL remains enabled during standby mode and continues to dissipate power.

```
            di                  /* Globally disable interrupts for PLL lock.      */
            pllc=0xa9f2         /* pllc[15]=1 enables the PLL to run at 100 MHz    */
                                /* with CKI=10 MHz. CKI must remain running.       */
pll_buzz:                       /* Assure time for PLL to lock                     */
            if lock goto select_pll
            goto pll_buzz
select_pll:
            pllc=0xe9f2         /* pllc[14]=1 selects the PLL.                     */
            ei                  /* Globally re-enable interrupts.                  */

                    .
                    .           /* user code with CLK = 100 MHz                    */
                    .
            pllc=0xa9f2         /* Prepare for standby - deselect PLL,             */
            powerc=0x1c1f       /* turn off peripherals and select slow clock.     */
            2*nop               /* Wait for it to take effect.                     */
            ioc=0x0040          /* Hold CKO low.                                   */
_standby:   alf=0x8000          /* Set AWAIT bit (stop core clock; interrupt logic */
            nop                 /* active.) nops needed for bedtime execution.     */
            nop                 /* Reduced standby power plus PLL power consumed.   */
            nop                 /* Interrupt wakes up the core.                    */
cont:       ...                 /* User code executes here.                        */
            pllc=0xe9f2         /* Reselect the PLL - PLL already locked.          */
            powerc=0x0000       /* Turn off slow clock and turn peripherals back on.*/
            2*nop               /* Wait for it to take effect.                     */
            ioc=0x0000          /* CKO is free-running.                            */
```

1. The program exits low-power standby mode when any enabled interrupt occurs. Therefore, it is assumed that interrupts are globally enabled and at least one interrupt is individually enabled.

Hardware Architecture (continued)

Power Management (continued)

Software Stop, PLL Enabled and Not Selected. The PLL is enabled to run at 100 MHz, assuming a constant CKI input clock of 10 MHz. Prior to performing a software stop by setting the NOCK bit (**powerc**[9]), the program reduces power by turning off all the peripherals and holding the CKO pin low. Setting the NOCK bit shuts off the internal clock and stops program execution until an interrupt on the INT0 pin restarts the internal clock. (Alternatively, INT1 or RSTB can be used to restart the clock.) The device restarts with CKI as the internal clock before the program reselects the PLL clock. The PLL remains enabled during software stop and continues to dissipate power.

```
            di                  /* Globally disable interrupts for PLL lock.    */
            pllc=0xa9f2         /* pllc[15]=1 enables the PLL to run at 100 MHz  */
                                /* with CKI=10 MHz. CKI must remain running.     */
pll_buzz:                       /* Assure time for PLL to lock                   */
            if lock goto select_pll
            goto pll_buzz
select_pll:
            pllc=0xe9f2         /* pllc[14]=1 selects the PLL.                   */
            ei                  /* Globally re-enable interrupts.                */
                .
                .               /* user code with CLK = 100 MHz                  */
                .
            pllc=0xa9f2         /* Prepare for stop--deselect PLL (select CKI),  */
            powerc=0x189f       /* set INT0EN, turn off peripherals.             */
            2*nop               /* Wait for it to take effect.                   */
            di
            inc0=NO_INT0        /* Disable the INT0 interrupt (Clear inc0[11:10]). */
            ei
            ioc = 0x0040        /* Hold CKO low.                                 */
_nock:      powerc=0x1e9f       /* Set NOCK to stop internal clock.              */
                                /* Minimum switching power consumed here.        */
            3*nop               /* Some nops are needed.                         */
                                /* INT0 pin clears the NOCK bit; clocking resumes. */
cont:       ...                 /* User code executes here.                      */
            powerc=0x0000       /* Clear INT0EN bit, select high-speed clock,    */
                                /* turn on peripherals                           */
            2*nop               /* Wait for it to take effect                    */
            pllc=0xe9f2         /* Reselect the PLL - PLL already locked.        */
            ins 0x0020          /* Clear the INT0 status bit.                    */
            di
            inc0=INT0           /* Safe to reenable the INT0 interrupt.          */
            ei
            ioc=0x0000          /* CKO is free-running PLL clock.                */
```

Hardware Architecture (continued)

Power Management (continued)

Software Stop, PLL Disabled and Not Selected. The PLL is enabled to run at 100 MHz, assuming a constant CKI input clock of 10 MHz. Prior to performing a software stop by setting the NOCK bit (**powerc**[9]), the program reduces power by turning off all the peripherals, holding the CKO pin low, and disabling the PLL. Because the PLL is disabled (powered down) during software stop, it does not dissipate power. The device restarts with CKI as the internal clock before the program reselects the PLL clock. After coming out of software stop, the program must enable the PLL and wait for it to lock before reselecting it.

```
            di                  /* Globally disable interrupts for PLL lock.    */
            pllc=0xa9f2         /* pllc[15]=1 enables the PLL to run at 100 MHz  */
                                /* with CKI=10 MHz. CKI must remain running.     */
pll_buzz:                       /* Assure time for PLL to lock                   */
            if lock goto select_pll
            goto pll_buzz
select_pll: pllc=0xe9f2         /* pllc[14]=1 selects the PLL.                   */
            ei                  /* Globally re-enable interrupts.                */
                .
                .               /* user code with CLK = 100 MHz                  */
                .
            pllc=0x29f2         /* Prepare for stop--deselect and disable PLL... */
            powerc=0x189f       /* (select CKI), set INT0EN, turn off peripherals. */
            2*nop               /* Wait for it to take effect.                   */
            di
            inc0=NO_INT0        /* Disable the INT0 interrupt (Clear inc0[11:10]). */
            ei
            ioc = 0x0040        /* Hold CKO low.                                 */
_nock:      powerc=0x1e9f       /* Set NOCK to stop internal clock.              */
                                /* Minimum switching power consumed here.        */
            3*nop               /* Some nops are needed.                         */
                                /* INT0 pin clears the NOCK bit; clocking resumes. */
cont:       ...                 /* User code executes here.                      */
            powerc=0x0000       /* Clear INT0EN bit, select high-speed clock,    */
                                /* turn on peripherals                           */
            2*nop               /* Wait for it to take effect                    */
            ins 0x0020          /* Clear the INT0 status bit.                    */
            di                  /* Globally disable interrupts for PLL lock...   */
            inc0=INT0           /* and inc0 change. Safe to reenable INT0.       */
            pllc=0xa9f2         /* pllc[15]=1 enables the PLL to run at 100 MHz. */
pll_buzz2:                      /* Assure time for PLL to lock                   */
            if lock goto select_pll2
            goto pll_buzz2
select_pll2:pllc=0xe9f2         /* pllc[14]=1 selects the PLL.                   */
            ei                  /* Globally re-enable interrupts.                */
            ioc=0x0000          /* CKO is free-running PLL clock.                */
```

Software Architecture

Instruction Set Quick Reference

The DSP16210 instruction set consists of both 16-bit and 32-bit wide instructions and resembles C-code. The following table defines the seven types of instructions. The assembler translates a code line into the most efficient instruction(s). See Table 33 on page 77 for instruction set notation conventions.

Table 31. DSP16210 Instruction Groups

Instruction Group	F Title (If Applicable)	Description
MAC	{F1} {TRANSFER} {F1E} {TRANSFER} {if CON} F1E	The powerful MAC instruction group is the primary group of instructions used for signal processing. Up to two data transfers can be combined with up to four parallel DAU operations in a single MAC instruction to execute simultaneously[†]. The DAU operation combinations include (but are not limited to) either a dual-MAC[‡] operation, an ALU operation and a BMU operation, or an ALU/ACS operation and an ADDER operation. The F1E instructions that do not include a transfer statement can execute conditionally based on the state of flags.
Special Function	{if CON} F2 ifc CON F2 {if CON} F2E ifc CON F2E	Special functions include rounding, negation, absolute value, and fixed arithmetic left and right shift operations. The operands are an accumulator, another DAU register, or an accumulator and another DAU register. Some special function instructions increment counters. Special functions execute conditionally based on the state of flags.
ALU	F3 {if CON} F3E	ALU instructions operate on two accumulators or on an accumulator and another DAU register. Many instructions can also operate on an accumulator and an immediate data word. The ALU operations are add, subtract, logical AND, logical OR, exclusive OR, maximum, minimum, and divide-step. Some F3E instructions include a parallel ADDER operation. The F3E instructions can execute conditionally based on the state of flags.
BMU	F4 {if CON} F4E	Full barrel shifting, exponent computation, normalization computation, bit-field extraction or insertion, and data shuffling between two accumulators are BMU operations that act on the accumulators. BMU operations are controlled by an accumulator, an auxiliary register, or a 16-bit immediate value. The F4E instructions can execute conditionally based on the state of flags.
Data Move and Pointer Arithmetic	—	Data move instructions transfer data between two registers or between a register and memory. This instruction group also supports immediate loads of registers, conditional register-to-register moves, pipeline block moves, and specialized stack operations. Pointer arithmetic instructions perform arithmetic on data pointers and do not perform a memory access.
Control	—	The control instruction group contains branch and call subroutine instructions with either a 20-bit absolute address or a 12-bit or 16-bit PC-relative address. This group also includes instructions to enable and disable interrupts. Some control instructions can execute conditionally based on the state of processor flags.
Cache	—	Cache instructions implement low-overhead loops by loading a set of up to 31 instructions into cache memory and repetitively executing them as many as $2^{16} - 1$ times.

† Executes in one instruction cycle in most cases.
‡ A dual-MAC operation consists of two multiplies and an add or subtract operation by the ALU, an add or subtract operation by the ADDER, or both.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

See the *DSP16000 Digital Signal Processor Core* Information Manual and *DSP16000 Digital Signal Processor Core Instruction Set Reference Manual* for a detailed description of:

- The Instruction Set
- Pipeline Hazards[1]
- Instruction Encoding Formats and Field Descriptions
- Instruction Set Reference

Table 32 starting on page 71 lists the entire instruction set with its cycle performance and the number of instruction/coefficient memory locations required for each. Below is an illustration of a single row of the table and a description of how to interpret its contents.

QUANTITY OF PROGRAM MEMORY
USED BY THE INSTRUCTION.
(EITHER 1 OR 2 16-bit words)

INSTRUCTIONS ARE GROUPED INTO
CATEGORIES (ONE OF SEVEN).

F TITLE
(IF APPLICABLE)

FLAGS AFFECTED BY
THIS INSTRUCTION[†]

Instruction	Flags	Cycles		Words
	szlme	Out	In	
ALU Group				
aD = aS OP ⟨aTE,pE⟩ (F3)	szlm–	1	1	1

INSTRUCTION SYNTAX

THE NUMBER OF INSTRUCTION CYCLES
USED WHEN THE INSTRUCTION IS EXE-
CUTED OUTSIDE OF THE CACHE.

THE NUMBER OF INSTRUCTION CYCLES
USED WHEN THE INSTRUCTION IS EXE-
CUTED INSIDE OF THE CACHE. A DASH
(—) INDICATES THE INSTRUCTION IS NOT
CACHABLE.

† szlme corresponds to the LMI (s), LEQ (z), LLV (l), LMV (m), and EPAR (e) flags. If a letter appears in this column, the corresponding flag is affected by this instruction. If a dash appears in this column, the corresponding flag is unaffected by this instruction. In the example shown, the instruction affects all flags except for EPAR. For MAC group instructions with both an ALU/ACS operation and an ADDER or BMU operation, the ALU/ACS result affects the LMI, LEQ, LLV, and LMV flags and the EPAR flag is unaffected. See Table 37 on page 85 for additional information.

Figure 22. Interpretation of the Instruction Set Summary Table

Table 33 on page 77 summarizes the instruction set notation conventions for interpreting the instruction syntax descriptions. Table 34 starting on page 78 is an overall replacement table that summarizes the replacement for every upper-case character string in the instruction set summary table (Table 32) except for F1 and F1E in the MAC instruction group. Table 35 on page 81 describes the replacement for the F1 field and Table 36 starting on page 83 describes the replacement for the F1E field.

1. A pipeline hazard occurs when a write to a register precedes an access that uses the same register and that register is not updated because of pipeline timing. The DSP16000 assembler automatically inserts a **nop** in this case to avoid the hazard.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 32. Instruction Set Summary

Instruction			Flags	Cycles		Words
			szlme	Out	In	
Multiply/Accumulate (MAC) Group						
⟨F1⟩	Y		szlm–	1	1	1
⟨F1⟩	x⟨h,l⟩=Y		szlm–			
⟨F1⟩	y⟨h,l⟩=Y		szlm–			
⟨F1⟩	aD̄⟨h,l⟩=Y		szlm–			
⟨F1⟩	Y=y⟨h,l⟩		szlm–			
⟨F1⟩	Y=aT⟨h,l⟩		szlm–			
⟨F1⟩ yh=aTh		xh=X	szlm–	1+Xc†		
⟨F1⟩	yh=Y	xh=X	szlm–			
⟨if CON⟩ F1E			szlme	1	1	2
⟨F1E⟩ y⟨h,l⟩=aTE⟨h,l⟩			szlme			
⟨F1E⟩ aTE⟨h,l⟩=y⟨h,l⟩			szlme			
⟨F1E⟩ y=aE_Ph			szlme			
⟨F1E⟩ aE_Ph=y			szlme			
⟨F1E⟩	x⟨h,l⟩=YE		szlme			
⟨F1E⟩	y⟨h,l⟩=YE		szlme			
⟨F1E⟩	aTE⟨h,l⟩=YE		szlme			
⟨F1E⟩	aE_Ph=YE		szlme			
⟨F1E⟩	YE=x⟨h,l⟩		szlme			
⟨F1E⟩	YE=y⟨h,l⟩		szlme			
⟨F1E⟩	YE=aTE⟨h,l⟩		szlme			
⟨F1E⟩	YE=aE_Ph		szlme			
⟨F1E⟩ y⟨h⟩=*r0 r0=rNE+jhb			szlme			
⟨F1E⟩	YE		szlme			
⟨F1E⟩		x⟨h,l⟩=XE	szlme	1+Xc†		
⟨F1E⟩		aTE⟨h,l⟩=XE	szlme			
⟨F1E⟩		aE_Ph=XE	szlme			
⟨F1E⟩ y=aE_Ph		x⟨h⟩=XE	szlme			
⟨F1E⟩ y⟨h⟩=aTE⟨h⟩		x⟨h⟩=XE	szlme			
⟨F1E⟩ aTE⟨h⟩=y⟨h⟩		x⟨h⟩=XE	szlme			
⟨F1E⟩	y⟨h⟩=YE‡	a4⟨h⟩=XE	szlme			
⟨F1E⟩	y⟨h,l⟩=YE	x⟨h⟩=XE	szlme			
⟨F1E⟩	YE=y⟨h,l⟩	x⟨h⟩=XE	szlme			
⟨F1E⟩	y⟨h⟩=YE‡	a4_5h=XE	szlme			
⟨F1E⟩	YE=a6_7h	x⟨h⟩=XE	szlme			
⟨F1E⟩	YE=a6⟨h⟩‡	x⟨h⟩=XE	szlme			
⟨F1E⟩	YE=a6⟨h⟩‡	a4⟨h⟩=XE	szlme			

† Xc is one cycle if XAAU contention occurs and zero cycles otherwise. XAAU contention occurs frequently for these instruction types and can only be avoided by use of the cache.

‡ For this transfer, the postincrement options ***rME** and ***rME−−** are not available for double-word loads.

§ The – (40-bit subtraction) operation is encoded as **aDE=aSE+IM16** with the IM16 value negated.

†† For conditional branch instructions, the execution time is two cycles if the branch is not taken.

‡‡ The instruction performs the same function whether or not ⟨**near**⟩ (optional) is included.

§§ Not including the N instructions.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 32. Instruction Set Summary (continued)

Instruction	Flags szlme	Cycles Out	Cycles In	Words
Multiply/Accumulate (MAC) Group (continued)				
⟨F1E⟩ y⟨h⟩=*r0 r0=rNE+j⟨lb⟩ j=k k=XE	szlme	$1+X_c$†	1	2
⟨F1E⟩ XE	szlme			
Special Function Group				
⟨if CON⟩ aD=aS>>⟨1,4,8,16⟩ (F2)	szlme	1	1	1
ifc CON aD=aS>>⟨1,4,8,16⟩ (F2)	szlme			
⟨if CON⟩ aD=aS (F2)	szlm–			
ifc CON aD=aS (F2)	szlm–			
⟨if CON⟩ aD=–aS (F2)	szlm–			
ifc CON aD=–aS (F2)	szlm–			
⟨if CON⟩ aD=~aS (F2)	szlm–			
ifc CON aD=~aS (F2)	szlm–			
⟨if CON⟩ aD=rnd(aS) (F2)	szlm–			
ifc CON aD=rnd(aS) (F2)	szlm–			
⟨if CON⟩ aDh=aSh+1 (F2)	szlm–			
ifc CON aDh=aSh+1 (F2)	szlm–			
⟨if CON⟩ aD=aS+1 (F2)	szlm–			
ifc CON aD=aS+1 (F2)	szlm–			
⟨if CON⟩ aD=⟨y,p0⟩ (F2)	szlm–			
ifc CON aD=⟨y,p0⟩ (F2)	szlm–			
⟨if CON⟩ aD=aS<<⟨1,4,8,16⟩ (F2)	szlme			
ifc CON aD=aS<<⟨1,4,8,16⟩ (F2)	szlme			
⟨if CON⟩ aDE=aSE>>⟨1,2,4,8,16⟩ (F2E)	szlme	1	1	2
ifc CON aDE=aSE>>⟨1,2,4,8,16⟩ (F2E)	szlme			
⟨if CON⟩ aDE=aSE (F2E)	szlm–			
ifc CON aDE=aSE (F2E)	szlm–			
⟨if CON⟩ aDE=–aSE (F2E)	szlm–			
ifc CON aDE=–aSE (F2E)	szlm–			
⟨if CON⟩ aDE=~aSE (F2E)	szlm–			
ifc CON aDE=~aSE (F2E)	szlm–			
⟨if CON⟩ aDE=rnd(⟨aSE,pE⟩) (F2E)	szlm–			
ifc CON aDE=rnd(⟨aSE,pE⟩) (F2E)	szlm–			
⟨if CON⟩ aDE=rnd(–pE) (F2E)	szlm–			
ifc CON aDE=rnd(–pE) (F2E)	szlm–			
⟨if CON⟩ aDE=rnd(aSE+pE) (F2E)	szlm–			
ifc CON aDE= rnd(aSE+pE) (F2E)	szlm–			
⟨if CON⟩ aDE=rnd(aSE–pE) (F2E)	szlm–			
ifc CON aDE=rnd(aSE–pE) (F2E)	szlm–			

† X_c is one cycle if XAAU contention occurs and zero cycles otherwise. XAAU contention occurs frequently for these instruction types and can only be avoided by use of the cache.

‡ For this transfer, the postincrement options ***rME** and ***rME– –** are not available for double-word loads.

§ The – (40-bit subtraction) operation is encoded as **aDE=aSE+IM16** with the IM16 value negated.

†† For conditional branch instructions, the execution time is two cycles if the branch is not taken.

‡‡ The instruction performs the same function whether or not ⟨**near**⟩ (optional) is included.

§§ Not including the N instructions.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 32. Instruction Set Summary (continued)

Instruction		Flags	Cycles		Words
		szlme	Out	In	
Special Function Group (continued)					
{if CON} aDE=abs(aSE)	(F2E)	szlm–	1	1	2
ifc CON aDE=abs(aSE)	(F2E)	szlm–			
{if CON} aDEh=aSEh+1	(F2E)	szlm–			
ifc CON aDEh=aSEh+1	(F2E)	szlm–			
{if CON} aDE=aSE+1	(F2E)	szlm–			
ifc CON aDE=aSE+1	(F2E)	szlm–			
{if CON} aDE=⟨y,pE⟩	(F2E)	szlm–			
ifc CON aDE=⟨y,pE⟩	(F2E)	szlm–			
{if CON} aDE=⟨−y,−pE⟩	(F2E)	szlm–			
ifc CON aDE=⟨−y,−pE⟩	(F2E)	szlm–			
{if CON} aDE=aSE<<⟨1,2,4,8,16⟩	(F2E)	szlme			
ifc CON aDE=aSE<<⟨1,2,4,8,16⟩	(F2E)	szlme			
ALU Group					
aD=aS OP ⟨aTE,pE⟩	(F3)	szlm–	1	1	1
aD=⟨aTE,pE⟩−aS	(F3)	szlm–			
aD=FUNC(aS,⟨aTE,pE⟩)	(F3)	szlm–			
aS−⟨aTE,pE⟩	(F3)	szlm–			
aS&⟨aTE,pE⟩	(F3)	szlm–			
{if CON} aDE=aSE OP ⟨pE,y⟩	(F3E)	szlm–	1	1	2
{if CON} aDE=aSE OP aTE	(F3E)	szlm–			
{if CON} aDE=⟨pE,y⟩−aSE	(F3E)	szlm–			
{if CON} aDE=FUNC(aSE,⟨pE,y⟩)	(F3E)	szlm–			
{if CON} aDE=FUNC(aSE,aTE)	(F3E)	szlm–			
{if CON} aSE−⟨pE,y⟩	(F3E)	szlm–			
{if CON} aSE&⟨pE,y⟩	(F3E)	szlm–			
{if CON} aSE−aTE	(F3E)	szlm–			
{if CON} aSE&aTE	(F3E)	szlm–			
{if CON} aDEE=aSEE±aTEE aDPE=aSPE±aTPE	(F3E)	szlm–			
if CON aDE=aSE+aTE else aDE=aSE−aTE	(F3E)	szlm–			
aDE=aSE⟨h,l⟩ OP IM16§	(F3 with immediate)	szlm–	1	1	2
aDE=IM16−aSE⟨h,l⟩	(F3 with immediate)	szlm–			
aSE⟨h,l⟩−IM16	(F3 with immediate)	szlm–			
aSE⟨h,l⟩&IM16	(F3 with immediate)	szlm–			

† X_C is one cycle if XAAU contention occurs and zero cycles otherwise. XAAU contention occurs frequently for these instruction types and can only be avoided by use of the cache.

‡ For this transfer, the postincrement options ***rME** and ***rME−−** are not available for double-word loads.

§ The − (40-bit subtraction) operation is encoded as **aDE=aSE+IM16** with the IM16 value negated.

†† For conditional branch instructions, the execution time is two cycles if the branch is not taken.

‡‡ The instruction performs the same function whether or not {**near**} (optional) is included.

§§ Not including the N instructions.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 32. Instruction Set Summary (continued)

Instruction		Flags szlme	Cycles Out	Cycles In	Words
BMU Group					
aD=aS SHIFT ⟨aTEh,arM⟩	(F4)	szlme	1	1	1
aDh=exp(aTE)	(F4)	szlme			
aD=norm(aS, ⟨aTEh,arM⟩)	(F4)	szlme			
aD=extracts(aS,aTEh) aD=extractz(aS,aTEh)	(F4)	szlme			
aD=inserts(aS,aTEh) aD=insertz(aS,aTEh)	(F4)	szlme			
aD=extract(aS,arM) aD=extracts(aS,arM) aD=extractz(aS,arM)	(F4)	szlme			
aD=insert(aS,arM) aD=inserts(aS,arM) aD=insertz(aS,arM)	(F4)	szlme			
aD=aS:aTE	(F4)	szlm–			
aDE=extract(aSE,IM8W,IM8O) aDE=extracts(aSE,IM8W,IM8O) aDE=extractz(aSE,IM8W,IM8O)	(F4 with immediate)	szlme	1	1	2
aDE=insert(aSE,IM8W,IM8O) aDE=inserts(aSE,IM8W,IM8O) aDE=insertz(aSE,IM8W,IM8O)	(F4 with immediate)	szlme			
aDE=aSE SHIFT IM16	(F4 with immediate)	szlme			
{if CON} aDE=aSE SHIFT⟨aTEh,arM⟩	(F4E)	szlme	1	1	2
{if CON} aDEh=exp(aTE)	(F4E)	szlme			
{if CON} aDE=norm(aSE,⟨aTEh,arM⟩)	(F4E)	szlme			
{if CON} aDE=extracts(aSE,aTEh) {if CON} aDE=extractz(aSE,aTEh)	(F4E)	szlme			
{if CON} aDE=inserts(aSE,aTEh) {if CON} aDE=insertz(aSE,aTEh)	(F4E)	szlme			
{if CON} aDE=extract(aSE,arM) {if CON} aDE=extracts(aSE,arM) {if CON} aDE=extractz(aSE,arM)	(F4E)	szlme			
{if CON} aDE=insert(aSE,arM) {if CON} aDE=inserts(aSE,arM) {if CON} aDE=insertz(aSE,arM)	(F4E)	szlme			
{if CON} aDE=aSE:aTE	(F4E)	szlm–			

† X_C is one cycle if XAAU contention occurs and zero cycles otherwise. XAAU contention occurs frequently for these instruction types and can only be avoided by use of the cache.

‡ For this transfer, the postincrement options ***rME** and ***rME−−** are not available for double-word loads.

§ The − (40-bit subtraction) operation is encoded as **aDE=aSE+IM16** with the IM16 value negated.

†† For conditional branch instructions, the execution time is two cycles if the branch is not taken.

‡‡ The instruction performs the same function whether or not {**near**} (optional) is included.

§§ Not including the N instructions.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 32. Instruction Set Summary (continued)

Instruction		Flags	Cycles		Words
		szlme	Out	In	
Data Move and Pointer Arithmetic Group					
RAB=IM20		—	1	1	2
RA=IM4		—	1	1	1
RA$_D$=RA$_S$		—	1	1	1
{if CON} RAB$_D$=RAB$_S$		—			2
RB=aTE⟨h,l⟩		—	1	1	1
aTE⟨h,l⟩=RB		—			
RA=Y		—	1	1	1
Y=RA		—			
RAB=YE		—	1	1	2
YE=RC		—			
RAB=*sp++2		—	1	1	1
*sp−−2=RC		—			
sp−−2		—			
*sp=RC		—			
push RC pop RC		—			
r3−−sizeof(RAB)		—			
RA=*(sp+IM5)		—	2	2	1
*(sp+IM5)=RA		—			
RAB=*(RP+IM12)		—	2	2	2
*(RP+IM12)=RC		—			
RAB=*(RP+⟨j,k⟩)		—			
*(RP+⟨j,k⟩)=RC		—			
RY=RP+IM12		—	1	1	2
RY=RP+⟨j,k⟩		—			
RAB=*r7	r7=sp+IM11	—	1	1	2
*r7=RC	r7=sp+IM11	—			
YE‡=x{h}	x{h}=XE	—	1+X$_C$†	1	2

† X$_C$ is one cycle if XAAU contention occurs and zero cycles otherwise. XAAU contention occurs frequently for these instruction types and can only be avoided by use of the cache.

‡ For this transfer, the postincrement options ***rME** and ***rME−−** are not available for double-word loads.

§ The − (40-bit subtraction) operation is encoded as **aDE=aSE+IM16** with the IM16 value negated.

†† For conditional branch instructions, the execution time is two cycles if the branch is not taken.

‡‡ The instruction performs the same function whether or not {**near**} (optional) is included.

§§ Not including the N instructions.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 32. Instruction Set Summary (continued)

Instruction	Flags	Cycles		Words
	szlme	Out	In	
Control Group				
{near} goto IM12‡‡	—	3	—	1
{near} call IM12‡‡	—			
{if CON} goto IM16	—	3††	—	2
{if CON} call IM16	—			
far goto IM20	—	3	—	
far call IM20	—			
{if CON} goto ptE	—	3††	—	1
{if CON} call ptE	—			
{if CON} call pr	—			
tcall	—	3	—	
icall IM6	—			
{if CON} return	—	3††	—	
ireturn	—	3	—	
treturn	—			
ei di	—	1	1	
Cache Group				
do K {N_INSTR}	—	1§§	—	1§§
redo K	—	2	—	1
do cloop {N_INSTR}	—	1§§	—	1§§
redo cloop	—	2	—	1

† X_C is one cycle if XAAU contention occurs and zero cycles otherwise. XAAU contention occurs frequently for these instruction types and can only be avoided by use of the cache.

‡ For this transfer, the postincrement options ***rME** and ***rME−−** are not available for double-word loads.

§ The − (40-bit subtraction) operation is encoded as **aDE=aSE+IM16** with the IM16 value negated.

†† For conditional branch instructions, the execution time is two cycles if the branch is not taken.

‡‡ The instruction performs the same function whether or not {**near**} (optional) is included.

§§ Not including the N instructions.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 33 defines the symbols used in instruction descriptions. Some symbols and characters are part of the instruction syntax, and must appear as shown within the instruction. Other symbols are representational and are replaced by other characters. The table groups these two types of symbols separately.

Table 33. Notation Conventions for Instruction Set Descriptions

Symbol		Meaning
Part of Syntax	*	16-bit x 16-bit multiplication resulting in a 32-bit product.
		Exception: When used as a prefix to an address register, denotes register-indirect addressing, e.g., ***r3**.
	**2	Squaring is a 16-bit x 16-bit multiplication of the operand with itself resulting in a 32-bit product.
	+	40-bit addition†.
	−	40-bit subtraction†.
	++	Register postincrement.
	−−	Register postdecrement.
	>>	Arithmetic right shift (with sign-extension from bit 39).
	<<	Arithmetic left shift (padded with zeros).
	>>>	Logical right shift (zero guard bits before shift).
	<<<	Logical left shift (padded with zeros; sign-extended from bit 31).
	&	40-bit bitwise logical AND†.
	|	40-bit bitwise logical OR†.
	^	40-bit bitwise logical exclusive-OR†.
	:	Register shuffle‡.
	~	One's complement (bitwise inverse).
	( )	Parentheses enclose multiple operands delimited by commas that are also part of the syntax.
	{ }	Braces enclose multiple instructions within a cache loop.
	_ (underscore)	The underscore character indicates an accumulator vector (concatenation of the high halves of a pair of sequential accumulators, e.g., **a0_1h**).
	lower-case	Lower-case characters appear as shown in the instruction.
Not Part of Syntax (Replaced)	⟨ ⟩	Angle brackets enclose items delimited by commas, one of which must be chosen.
	{ }	Mid braces enclose one or more optional items delimited by commas.
	±	Replaced by either + or −.
	UPPER-CASE	Upper-case characters, character strings, and characters plus numerals (e.g., M, CON, and IM16) are replaced. Replacement tables accompany each instruction group description.
	F Titles	Represents a statement of a DAU function:
			F1	MAC.
			F1E	Extended MAC.
			F2	Special function.
			F2E	Extended special function.
			F3	ALU.
			F3E	Extended ALU.
			F4	BMU.
			F4E	Extended BMU.

† The ALU/ACS and ADDER perform 40-bit operations, but the operands can be 16 bits, 32 bits, or 40 bits. In the special case of the split-mode F1E instruction (**xh=aSPEh±yh xl=aSPEl±yl aDE=aSEE+p0+p1 p0=xh**2 p1=xl**2**), the ALU performs two 16-bit addition/subtraction operations in parallel.
‡ Note that this symbol does not denote compound addressing as it does for the DSP16XX family.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 34. Overall Replacement Table

Symbol	Used in Instruction Type(s)	Replaced By	Description
aD	F1, F2, F3, F4	**a0** or **a1** (DSP16XX-compatible)	**D** indicates **destination** of an operation.
aS			**S** indicates **source** of an operation.
aT	F1		**T** indicates an accumulator that is the source of a data transfer.
a$\overline{\text{D}}$			$\overline{\text{D}}$ indicates the inverse of the destination.
aDE	F1E, F2E, F3/E, F4/E	**a0**, **a1**, **a2**, **a3**, **a4**, **a5**, **a6**, or **a7**	**D** indicates **destination** of an operation. **S** indicates **source** of an operation. **T** indicates an accumulator that is either an additional source for an operation or the source or destination of a data transfer. **E** indicates the **extended** set of accumulators.
aSE			
aTE	F1E, F3/E, F4/E, data move		
aDEE	F1E, F3E	a⟨DPE − 1⟩ → **a0**, **a2**, **a4**, or **a6**	**D** indicates **destination** of an operation. **S** indicates **source** of an operation. **T** indicates an accumulator that is either an additional source for an operation or the source or destination of a data transfer. The first **E** indicates an **even** accumulator that is paired with its corresponding paired extended (odd) accumulator, i.e., the matching **aDPE**, **aSPE**, or **aTPE** accumulator. The second **E** indicates the **extended** set of accumulators.
aSEE		a⟨SPE − 1⟩ → **a0**, **a2**, **a4**, or **a6**	
aTEE	F3E	a⟨TPE − 1⟩ → **a0**, **a2**, **a4**, or **a6**	
aDPE	F1E, F3E	a⟨DEE + 1⟩ → **a1**, **a3**, **a5**, or **a7**	**P** indicates an odd accumulator that is **paired** with an even extended accumulator, i.e., the matching **aDEE**, **aSEE**, or **aTEE** accumulator. **E** indicates the **extended** set of accumulators.
aSPE		a⟨SEE + 1⟩ → **a1**, **a3**, **a5**, or **a7**	
aTPE	F3E	a⟨TEE + 1⟩ → **a1**, **a3**, **a5**, or **a7**	
aE_Ph	F1E	**a0_1h**, **a2_3h**, **a4_5h**, or **a6_7h**	An accumulator vector, i.e., the concatenated 16-bit high halves of two adjacent accumulators to form a 32-bit vector.
arM	F4, F4E	**ar0**, **ar1**, **ar2**, or **ar3**	One of the four auxiliary accumulators.
cloop	cache	11-bit unsigned value (1 to 65,535)	16-bit value that specifies the number of times the instructions execute.
CON	F1E, F2, F2E, F3E, F4E, control, data move	**mi**, **pl**, **eq**, **ne**, **lvs**, **lvc**, **mvs**, **mvc**, **heads**, **tails**, **c0ge**, **c0lt**, **c1ge**, **c1lt**, **true**, **false**, **gt**, **le**, **oddp**, **evenp**, **smvs**, **smvc**, **jobf**, **jibe**, **jcont**, **lock**, **mbusy1**, **mbusy0**, **somef**, **somet**, **allf**, or **allt**	Conditional mnemonics. Certain instructions are conditionally executed, e.g., **if CON F2E**.
FUNC	F3, F3E	**max**, **min**, or **divs**	One of three ALU functions: maximum, minimum, or divide-step.
IM4	data move	4-bit unsigned immediate value (0 to 15)	Signed/unsigned status of the IM4 value matches that of the destination register of the data move assignment instruction.
		4-bit signed immediate value (−8 to +7)	
IM5	data move	5-bit unsigned immediate value (0 to 31)	Added to stack pointer **sp** to form stack address.
IM6	control	6-bit unsigned immediate value (0 to 63)	Vector for **icall** instruction.
IM8O IM8W	F4	8-bit unsigned immediate value (0 to 255)	Offset and width for bit-field insert and extract instructions. The BMU truncates these values to 6 bits.
IM11	data move	11-bit unsigned immediate value (0 to 2047)	Added to stack pointer **sp** to form stack address.

† The size of the transfer (single- or double-word) depends on the size of the register on the other side of the equal sign.
‡ These postmodification options are not available for a double-word load except for a load of an accumulator vector.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 34. Overall Replacement Table (continued)

Symbol	Used in Instruction Type(s)	Replaced By		Description
IM12	control	12-bit signed immediate value (−2048 to +2047)		PC-relative near address for **goto** and **call** instructions.
	data move and pointer arithmetic			Postmodification to a general YAAU pointer register to form address for data move.
				Added to the value of a general YAAU pointer register, and the result is stored into any YAAU register.
IM16	control	16-bit signed immediate value (−32,768 to +32,767)		Offset for conditional PC-relative **goto/call** instructions.
	F3, F4			Operand for ALU or BMU operation.
IM20	control, data move	20-bit unsigned immediate value (0 to 1,048,576)		Absolute (unsigned) far address for **goto** and **call** instructions. For data move instructions, the signed/unsigned status of the IM20 value matches that of the destination register of the assignment instruction.
		20-bit signed immediate value (−524,288 to 524,287)		
K	cache	1 to 127 or the value in **cloop**		For the **do K {N_INSTR}** and **redo K** cache instructions.
N		1 to 31		
OP	F1, F1E, F3, F3E	**+**, **−**, **&**, **|**, or **^**		40-bit ALU operation.
pE	F2E, F3, F3E	**p0** or **p1**		One of the product registers as source for a special function or ALU operation.
ptE	F1E, control, data move	**pt0** or **pt1**		One of the two XAAU pointer registers as address for an XE memory access (see XE entry in this table).
RA, RA_D, RA_S	data move	**a0**, **a1**, **a2**, **a3**, **a4**, **a5**, **a6**, **a7**, **a0h**, **a1h**, **a2h**, **a3h**, **a4h**, **a5h**, **a6h**, **a7h**, **a0l**, **a1l**, **a2l**, **a3l**, **a4l**, **a5l**, **a6l**, **a7l**, **alf**, **auc0**, **c0**, **c1**, **c2**, **h**, **i**, **j**, **k**, **p0**, **p0h**, **p0l**, **p1**, **p1h**, **p1l**, **pr**, **psw0**, **pt0**, **pt1**, **r0**, **r1**, **r2**, **r3**, **r4**, **r5**, **r6**, **r7**, **rb0**, **rb1**, **re0**, **re1**, **sp**, **x**, **xh**, **xl**, **y**, **yh**, or **yl**		One of the main set of core registers that is specified as the source or destination of a data move operation. The subscripts are used to indicate that two different registers can be specified, e.g., **RA_D = RA_S** describes a register-to-register move instruction where RA_D and RA_S are, in general, two different registers.
RB		core	**a0g**, **a1g**, **a2g**, **a3g**, **a4g**, **a5g**, **a6g**, **a7g**, **a0_1h**, **a2_3h**, **a4_5h**, **a6_7h**, **ar0**, **ar1**, **ar2**, **ar3**, **auc1**, **cloop**, **cstate**, **csave**, **inc0**, **inc1**, **ins**, **pi**, **psw1**, **ptrap**, **vbase**, or **vsw**	One of the secondary set of registers that is specified as the source or destination of a data move operation. This set includes core and off-core registers.
		off-core	**cbit**, **ioc**, **jiob**, **mcmd0**, **mcmd1**, **miwp0**, **miwp1**, **morp0**, **morp1**, **mwait**, **pllc**, **powerc**, **sbit**, **timer0**, **timer0c**, **timer1**, or **timer1c**	
RAB, RAB_D, RAB_S		Any of the RA or RB registers (see rows above)		Any one of the registers in the main (RA) or secondary (RB) sets of registers that is specified as the source or destination of a data move operation. The subscripts are used to indicate that two different registers can be specified.
RC		Any of the RA registers or any of the core RB registers (see rows above)		Any core register that is specified as the source or destination of a data move operation.
rM	F1, data move	**r0**, **r1**, **r2**, or **r3**		One of four general YAAU pointer registers used for a Y memory access (see Y entry in this table).

† The size of the transfer (single- or double-word) depends on the size of the register on the other side of the equal sign.
‡ These postmodification options are not available for a double-word load except for a load of an accumulator vector.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 34. Overall Replacement Table (continued)

Symbol	Used in Instruction Type(s)	Replaced By	Description
rME	F1E, data move	**r0**, **r1**, **r2**, **r3**, **r4**, **r5**, **r6**, or **r7**	One of eight general YAAU pointer registers used for a YE memory access (see YE entry in this table). **E** indicates the **extended** set of pointer registers.
rNE	F1E	**r1**, **r2**, **r3**, **r4**, **r5**, **r6**, or **r7**	One of seven general YAAU pointer registers used for a table look-up pointer update.
RP	data move and pointer arithmetic	**r0**, **r1**, **r2**, **r3**, **r4**, **r5**, **r6**, or **sp**	One of seven general YAAU pointer registers or the YAAU stack pointer.
RY		**r0**, **r1**, **r2**, **r3**, **r4**, **r5**, **r6**, **r7**, **sp**, **rb0**, **rb1**, **re0**, **re1**, **j**, or **k**	Any one of the YAAU registers, including the stack pointer, circular buffer pointers, and increment registers.
X	F1	***pt0++** or ***pt0++i**	A single-word location pointed to by **pt0**.
Y	F1	***rM**, ***rM++**, ***rM−−**, or ***rM++**j	A single-word location pointed to by **rM**.
	{F1} Y	**rM++**, **rM−−**, or **rM++**j	Modification of **rM** pointer register (no memory access).
	data move	***rM**, ***rM++**, ***rM−−**, or ***rM++**j	A single- or double-word† location pointed to by **rM**.
XE	F1E, data move	***ptE**‡, ***ptE++**, ***ptE−−**‡, ***ptE++h**, or ***ptE++i**	A single-word or double-word† memory location pointed to by **ptE**.
	{F1E} XE	**ptE++**, **ptE−−**, **ptE++h**, **ptE++i**, or **ptE++2**	Modification of **ptE** pointer register (no memory access).
YE	F1E, data move	***rME**, ***rME++**, ***rME−−**, ***rME++j**, or ***rME++k**	A single-word or double-word† memory location pointed to by **rME**.
	{F1E} YE	**rME++**, **rME−−**, **rME++j**, **rME++k**, **rME++2**, or **rME−−2**	Modification of **rME** pointer register (no memory access).

† The size of the transfer (single- or double-word) depends on the size of the register on the other side of the equal sign.
‡ These postmodification options are not available for a double-word load except for a load of an accumulator vector.

Table 35 on page 81 defines the F1 instruction syntax as any function statement combined with any transfer statement. Two types of F1 function statements are shown: the MAC (multiply/accumulate) type and the arithmetic/logic type. The MAC type is formed by combining any two items from the designated ALU and Multiplier columns. The arithmetic/logic type is chosen from the items in the designated Arithmetic/Logic Function Statement column.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 35. F1 Instruction Syntax

Combine Any F1 Function Statement with Any Transfer Statement				
F1 MAC Function Statement— **Combine Any Items in Following Two Columns:**		**Transfer Statement**	**Cycles (Out/In Cache)**[†]	**16-Bit Words**
ALU	**Multiplier**			
$aD = \{aS \pm\}$ p0	p0 = xh * yh	Y[‡]	1/1	1
(no ALU operation)[§]	(no multiply operation)[§]	$\langle x, y, a\overline{D}$[††]$\rangle\langle h, l\rangle = Y$	1/1	
F1 Arithmetic/Logic Function Statement (ALU)		$Y = \langle y, aT\rangle\langle h, l\rangle$	1/1	
$aD = \{aS\ OP\}$ y		$yh = \langle Y, aTh\rangle \qquad xh = X$	$1 + Xc$[‡‡]$/1$	
$aS - y$		(no transfer)[§§]	1/1	
aS & y				
nop[†††]				
(no F1 function statement)[§]				

† Not including conflict, misalignment, or external wait-states (see the *DSP16000 Digital Signal Processor Core* Information Manual).

‡ This Y transfer statement must increment or decrement the contents of an **rM** register. It is not necessary to include the * before the **rM** register because no access is made to a memory location.

§ Leave the ALU column blank to specify no ALU operation, the multiplier column blank to specify no multiply operation, or both columns blank to specify no F1 function statement. If both columns are left blank and a transfer statement is used (a transfer-only F1 instruction, i.e., **yh = *r2   xh = *pt0++**), the assembler interprets the F1 function statement as a **nop**.

†† For this instruction, **a$\overline{D}$** must be the opposite of **aD**, e.g., if **aD** is **a0**, **a$\overline{D}$** must be **a1** and vice versa.

‡‡ Xc is one cycle if XAAU contention occurs and zero cycles otherwise. XAAU contention occurs frequently for these instruction types and can only be avoided by use of the cache. See the *DSP16000 Digital Signal Processor Core* Information Manual.

§§ The assembler encodes an instruction that consists of a function statement **F1** with no transfer statement as **F1 *r0**.

††† **nop** is no-operation. A programmer can write **nop** with or without an accompanying transfer statement. The assembler encodes **nop** without a transfer statement as **nop *r0**.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 36 starting on page 83 summarizes the syntax for F1E function statements and the following paragraphs describe each class of instruction.

Note: Each function statement can be combined with a parallel transfer statement to form a single DSP16210 instruction.

General-Purpose MAC	Combine any ALU, ADDER, or ALU and ADDER operation from the left column with any single- or dual-multiply operation from the right column. Either column can be left blank.[1]
Additional General-Purpose MAC	These statements are general-purpose. The combinations of operations must be as shown. The first statement clears two accumulators and both product registers. The second statement is the equivalent of the F1 statement **aD=p0 p0=xh*yh** except that any accumulator aDE can be specified. The third statement is the equivalent of the F1 statement **aD=p0** except that any accumulator aDE can be specified. The fourth statement is a no-operation and, as with all F1E function statements, can be combined with a transfer statement.
Special-Purpose MAC for Mixed Precision	Combine any ADDER operation or any ALU and ADDER operation from the left column with any dual-multiply operation from the right column. Either column can be left blank.[1] These statements are intended for, but are not limited to, mixed-precision MAC applications. Mixed-precision multiplication is 16 bits x 31 bits.
Special-Purpose MAC for Double Precision	These statements are intended for, but are not limited to, double-precision MAC applications. The combinations of operations must be as shown. Double-precision multiplication is 31 bits x 31 bits.
Special-Purpose MAC for Viterbi	These statements are intended for, but are not limited to, Viterbi decoding applications. The combinations of operations must be as shown. This group includes ALU split-mode operations.
Special-Purpose MAC for FFT	This statement is intended for, but is not limited to, FFT applications.
ALU	These statements are ALU operations. The first three statements in this group are the equivalent of the F1 arithmetic/logic function statements.
Special-Purpose ALU/ACS, ADDER for Viterbi	These statements are intended for, but are not limited to, Viterbi decoding applications. They provide an ALU/ACS operation with or without a parallel ADDER operation. The combinations of operations must be as shown. This group includes the Viterbi compare functions.
Special-Purpose ALU, BMU	These statements are intended for, but are not limited to, special-purpose applications. They provide a BMU operation with or without a parallel ALU operation. The combinations of operations must be as shown.

---

1. If both columns are left blank and a transfer statement is used, the DSP16000 assembler interprets the F1E function statement as a no-operation (**nop**).

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 36. F1E Function Statement Syntax

General-Purpose MAC Function Statements—Combine Any Items in Two Columns:

ALU†	ADDER†	Multipliers	
aDE=aSE±p0		p0=xh*yh	
	aDE=aSE±p0±p1‡	p0=xh*yh	p1=xl*yl
aDEE=aSEE±p0	aDPE=aSPE±p1	p0=xh*yl	p1=xl*yh
(no ALU/ACS or ADDER operation)		p0=xh*yh	p1=xh*yl
		p0=xl*yh	p1=xl*yl
		(no multiply operation)	

Additional General-Purpose MAC Function Statements

ALU†	ADDER†	Multipliers	
aDE=0	aSE=0	p0=0	p1=0
aDE=p0		p0=xh*yh	
aDE=p0			
nop			

Special-Purpose MAC Function Statements for Mixed Precision—Combine Any Items in Two Columns:

ALU†	ADDER†	Multipliers	
	aDE=p0+(p1>>15)§	p0=xh*yh	p1=xh*(yl>>>1)
aDEE=aSE+aDPE	aDPE=p0+(p1>>15)§	p0=xl*yh	p1=xl*(yl>>>1)
(no ALU/ACS or ADDER operation)		(no multiply operation)	

Special-Purpose MAC Function Statements for Double Precision

ALU†	ADDER†	Multipliers	
	aDE=aSE+p0+(p1>>15)‡§	p0=xh*yh	p1=xh*(yl>>>1)
	aDE=aSE+p0+(p1>>15)‡§		
	aDE=p0+(p1>>15)§	p0=0	p1=(xl>>>1)*yh
aDEE=aSE+aDPE	aDPE=p0+(p1>>15)§	p0=0	p1=(xl>>>1)*yh
	aDE=(p0>>1)+(p1>>16)	p0=(xl>>>1)*yh	p1=xh*yh
aDEE=aSE+aDPE	aDPE=(p0>>1)+(p1>>16)	p0=(xl>>>1)*yh	p1=xh*yh
aDE=aSE+(p0>>1)			
		p0=xh*(yl>>>1)	p1=(xl>>>1)*(yl>>>1)
aDE=(aSE>>14)+p1		p0=xh*(yl>>>1)	p1=(xl>>>1)*(yl>>>1)
aDE=(aSE>>14)+p1			

† DAU flags are affected by the ALU or ALU/ACS operation (except for the split-mode function which does not affect the flags). If there is no ALU or ALU/ACS operation, the DAU flags are affected by the ADDER or BMU operation.

‡ If **auc0**[10] (FSAT field) is set, the result of the add/subtract of the first two operands is saturated to 32 bits prior to adding/subtracting the third operand and the final result is saturated to 32 bits.

§ If **auc0**[9] = 1, the least significant bit of **p1>>15** is cleared.

†† This split-mode instruction does not affect the DAU flags. Do not set FSAT for this instruction because if FSAT is set, the entire 32 bits are saturated.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Table 36. F1E Function Statement Syntax (continued)

Special-Purpose MAC Function Statements for Viterbi

ALU[†]		ADDER[†]	Multipliers	
xh=aSPEh+yh	xl=aSPEl+yl[††]	aDE=aSEE+p0+p1	p0=xh**2	p1=xl**2
xh=aSPEh−yh	xl=aSPEl−yl[††]	aDE=aSEE+p0+p1	p0=xh**2	p1=xl**2
		aDE=aSE+p0+p1[‡]	p0=xh**2	p1=xl**2

Special-Purpose MAC Function Statement for FFT

ALU[†]	ADDER[†]	Multipliers	
aDEE=−aSEE+p0	aDPE=−aSPE+p1	p0=xh*yh	p1=xl*yl

ALU Function Statements

aDE={aSE OP}y
aSE−y
aSE&y
aDE=aDE±aSE

Special-Purpose ALU/ACS, ADDER Function Statements for Viterbi

ALU/ACS[†]	ADDER[†]
aDEE=cmp0(aSEE,aDEE)	aDPE=aDPE+aSPE
aDE=cmp0(aSE,aDE)	
aDEE=cmp1(aSE,aDEE)	aDPE=aDEE−aSE
aDE=cmp1(aSE,aDE)	
aDEE=cmp2(aSE,aDEE)	aDPE=aDEE−aSE
aDE=cmp2(aSE,aDE)	
aDEE=aSEE+y	aDPE=aSPE−y
aDEE=aSEE−y	aDPE=aSPE+y

Special-Purpose ALU, BMU Function Statements

ALU[†]	BMU[†]
aDEE=rnd(aDPE)	aDPE=aSEE>>aSPEh
	aDE=aSEE>>aSPEh
aDE=abs(aDE)	aSE=aSE<<ar3
	aDE=aSE<<ar3
	aDE=aSE<<<ar3
aDEE=min(aDPE,aDEE)	aDPEh=exp(aSE)

† DAU flags are affected by the ALU or ALU/ACS operation (except for the split-mode function which does not affect the flags). If there is no ALU or ALU/ACS operation, the DAU flags are affected by the ADDER or BMU operation.

‡ If **auc0**[10] (FSAT field) is set, the result of the add/subtract of the first two operands is saturated to 32 bits prior to adding/subtracting the third operand and the final result is saturated to 32 bits.

§ If **auc0**[9] = 1, the least significant bit of **p1>>15** is cleared.

†† This split-mode instruction does not affect the DAU flags. Do not set FSAT for this instruction because if FSAT is set, the entire 32 bits are saturated.

Software Architecture (continued)

Instruction Set Quick Reference (continued)

Conditions Based on the State of Flags

A conditional instruction begins with either **if CON** or **ifc CON** where a condition to test replaces **CON**. Table 37 describes the complete set of condition codes available for use in conditional instructions. It also includes the state of the internal flag or flags that cause the condition to be true.

Table 37. DSP16210 Conditional Mnemonics

CON Encoding	CON Mnemonic	Flag(s) If CON Is True	Type[†]	Description
00000	mi	LMI = 1	Core	Most recent DAU result is negative.
00001	pl	LMI ≠ 1	Core	Most recent DAU result is positive or zero.
00010	eq	LEQ = 1	Core	Most recent DAU result is equal to zero.
00011	ne	LEQ ≠ 1	Core	Most recent DAU result is not equal to zero.
00100	lvs	LLV = 1	Core	Most recent DAU result has overflowed 40 bits.
00101	lvc	LLV ≠ 1	Core	Most recent DAU result has not overflowed 40 bits.
00110	mvs	LMV = 1	Core	Most recent DAU result has overflowed 32 bits.
00111	mvc	LMV ≠ 1	Core	Most recent DAU result has not overflowed 32 bits.
01000	heads	—	Core	Pseudorandom sequence generator output is set.
01001	tails	—	Core	Pseudorandom bit is clear.
01010	c0ge[‡]	—	Core	Current value in counter **c0** is greater than or equal to zero.
01011	c0lt[‡]	—	Core	Current value in counter **c0** is less than zero.
01100	c1ge[‡]	—	Core	Current value in counter **c1** is greater than or equal to zero.
01101	c1lt[‡]	—	Core	Current value in counter **c1** is less than zero.
01110	true	1	Core	Always.
01111	false	0	Core	Never.
10000	gt	(LMI ≠ 1) and (LEQ ≠ 1)	Core	Most recent DAU result is greater than zero.
10001	le	(LMI = 1) or (LEQ = 1)	Core	Most recent DAU result is less than or equal to zero.
10010	smvs	SLMV = 1	Core	A previous result has overflowed 32 bits (sticky flag).
10011	smvc	SLMV ≠ 1	Core	A previous result has not overflowed 32 bits since SLMV last cleared.
10100	oddp	EPAR ≠ 1	Core	Most recent 40-bit BMU result has odd parity.
10101	evenp	EPAR = 1	Core	Most recent 40-bit BMU result has even parity.
10110	jobf	JOBF = 1	JTAG	**jiob** output buffer full.
10111	jibe	JIBE = 1	JTAG	**jiob** input buffer empty.
11000	jcont	JCONT = 1	JTAG	JTAG continue.
11001	lock	LOCK = 1	CLOCK	PLL is locked.
11010	mbusy1	MBUSY1 = 1	MIOU1	MIOU1 has unfinished output pending.
11011	mbusy0	MBUSY0 = 1	MIOU0	MIOU0 has unfinished output pending.
11100	somef	SOMEF = 1	BIO	Some false (some tested input bits do not match the pattern).
11101	somet	SOMET = 1	BIO	Some true (some tested input bits match the pattern).
11110	allf	ALLF = 1	BIO	All false (all tested input bits do not match the pattern).
11111	allt	ALLT = 1	BIO	All true (all tested input bits match the pattern).

† All peripheral (off-core) flags are accessible in the **alf** register.
‡ Each test of **c0ge** or **c0lt** causes counter **c0** to postincrement. Each test of **c1ge** or **c1lt** causes counter **c1** to postincrement.

Software Architecture (continued)

Registers

Peripheral Register Write-Read Latency

As a consequence of the pipelined IDB, there is a write-to-read latency for peripheral (off-core) registers. This latency is automatically compensated by the DSP16000 assembler.

For all peripheral registers except MIOU registers, there is a one cycle write-to-read latency. For example:

```
timer0c=0x00aa   // update timer0c
nop              // inserted by assembler
a0h=timer0c      // returns 0x00aa
```

In the above example, the **nop** instruction (or any other instruction that does not read **timer0c**) is needed to guarantee that the subsequent read of **timer0c** returns the updated value. To prevent the assembler from inserting the **nop**, the programmer can insert any instruction.

For MIOU registers, there is a two instruction cycle latency before the most recently written MIOU register (**miwp⟨0—1⟩** or **morp⟨0—1⟩**) is returned by a subsequent peripheral register read. The assembler automatically inserts one or two nop instructions, as needed. See the program example below:

```
morp0=0x00aa    // update morp0
2*nop           // inserted by assembler
a3l=morp0       // returns 0x00aa
miwp1=a2h       // update miwp1
a2l=0x1234      // 1-cycle instruction
nop             // inserted by assembler
ar0=miwp1
```

Register Overview

DSP16210 registers fall into one of the following categories:

- Directly program-accessible (or register-mapped) registers are directly accessible in instructions and are designated with lower-case bold, e.g., **timer0**. These registers are summarized in Figure 23 on page 87 and in Table 38 starting on page 88.
- ESIO memory-mapped registers are designated with upper-case bold, e.g., **ICR**. These registers are summarized in Table 39 on page 90.
- MIOU-accessible registers are accessible only by MIOU commands, i.e., by writing the **mcmd0** or **mcmd1** register, and are designated with upper-case bold, e.g., **IBAS0**. These registers are summarized in Table 40 on page 90.
- DMA-accessible registers are SSIO or PHIF16 data registers that are accessible only via MIOU DMA in IORAM locations and are designated with upper-case bold, e.g., **PDX**(in). These registers are summarized in Table 41 on page 90.

Note: The program counter (**PC**) is an addressing register not accessible to the programmer or through external pins. The device automatically controls this register to properly sequence the instructions.

Figure 23 on page 87 depicts the directly program-accessible registers of which there are three types:

Data registers store data either from the result of instruction execution or from memory. Data registers become source operands for instructions. This class of registers also includes postincrement registers whose contents are added to address registers to form new addresses.

Control and Status registers are used to determine the state of the machine or to set different configurations to control the machine.

Address registers are used to hold memory location pointers. In some cases, the user can treat address registers as general-purpose data registers accessible by data move instructions.

Software Architecture (continued)

Registers (continued)
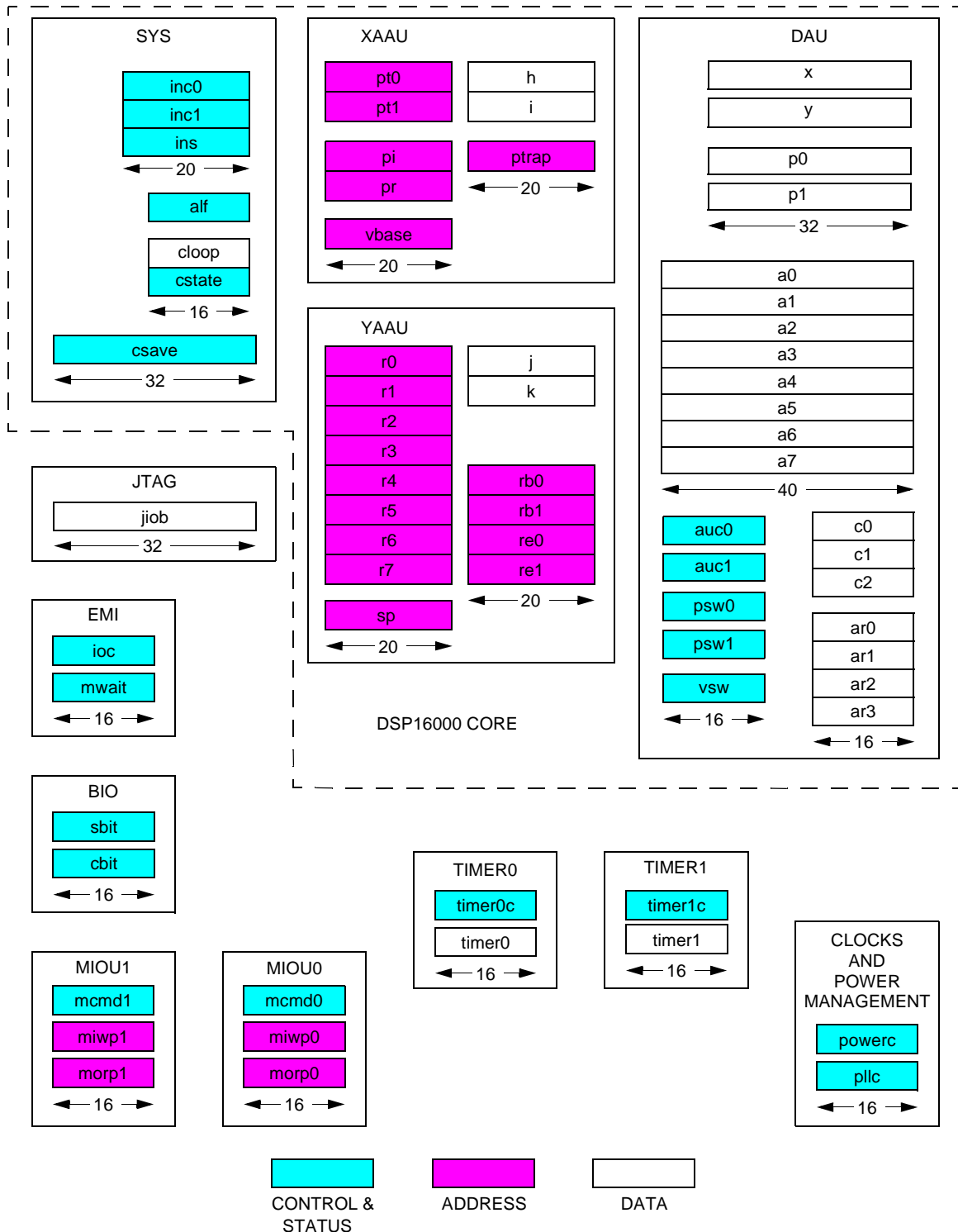
Register Overview (continued)

Figure 23. DSP16210 Program-Accessible Registers

Software Architecture (continued)

Registers (continued)

Register Overview (continued)

Table 38 lists all valid register designators as they appear in an instruction syntax. The table specifies a register's size, whether a register is readable or writable, a register's type, whether a register is signed or unsigned, and the hardware function block in which a register is located.

Table 38. Program-Accessible Registers by Type, Listed Alphabetically

Register Name	Description	Size (bits)	R/W†	Type‡	Signed§/ Unsigned	Function Block
a0, a1, a2, a3, a4, a5, a6, a7	Accumulators 0—7	40	R/W	data	signed	DAU
a0h, a1h, a2h, a3h, a4h, a5h, a6h, a7h	Accumulators 0—7, high halves (bits 31—16)	16	R/W	data	signed	DAU
a0l, a1l, a2l, a3l, a4l, a5l, a6l, a7l	Accumulators 0—7, low halves (bits 15—0)	16	R/W	data	signed	DAU
a0g, a1g, a2g, a3g, a4g, a5g, a6g, a7g	Accumulators 0—7, guard bits (bits 39—32)	8	R/W	data	signed	DAU
a0_1h, a2_3h, a4_5h, a6_7h	Accumulator vectors (concatenated high halves of two adjacent accumulators)	32	R/W	data	signed	DAU
alf	AWAIT and flags	16	R/W	c & s	unsigned	SYS
ar0, ar1, ar2, ar3	Auxiliary registers 0—3	16	R/W	data	signed	DAU
auc0, auc1	Arithmetic unit control	16	R/W	c & s	unsigned	DAU
c0, c1	Counters 0 and 1	16	R/W	data	signed	DAU
c2	Counter holding	16	R/W	data	signed	DAU
cbit	BIO control	16	R/W	control	unsigned	BIO
cloop	Cache loop count	16	R/W	data	unsigned	SYS
csave	Cache save	32	R/W	control	unsigned	SYS
cstate	Cache state	16	R/W	control	unsigned	SYS
h	Pointer postincrement	20	R/W	data	signed	XAAU
i	Pointer postincrement	20	R/W	data	signed	XAAU
ioc	I/O configuration	16	R/W	control	unsigned	EMI
inc0, inc1	Interrupt control 0 and 1	20	R/W	control	unsigned	SYS
ins	Interrupt status	20	R/W	status	unsigned	SYS
j	Pointer postincrement/offset	20	R/W	data	signed	YAAU
jhb	High byte of **j** (bits 15—8)	8	R	data	unsigned	YAAU
jlb	Low byte of **j** (bits 7—0)	8	R	data	unsigned	YAAU
jiob	JTAG test	32	R/W	data	unsigned	JTAG
k	Pointer postincrement/offset	20	R/W	data	signed	YAAU
mcmd0, mcmd1	MIOU command registers 0 and 1	16	W	control	unsigned	MIOU
miwp0, miwp1	MIOU IORAM input write pointers 0 and 1	16	R/W	address	unsigned	MIOU
morp0, morp1	MIOU IORAM output read pointers 0 and 1	16	R/W	address	unsigned	MIOU

† R indicates that the register is readable by instructions; W indicates the register is writable by instructions.
‡ c & s means control and status.
§ Signed registers are in two's complement format.
†† Some bits in the **psw0** and **psw1** registers are read only (writes to these bits are ignored).

Software Architecture (continued)

Registers (continued)

Register Overview (continued)

Table 38. Program-Accessible Registers by Type, Listed Alphabetically (continued)

Register Name	Description	Size (bits)	R/W[†]	Type[‡]	Signed[§]/ Unsigned	Function Block
mwait	EMI configuration	16	R/W	control	unsigned	EMI
p0	Product 0	32	R/W	data	signed	DAU
p0h	High half of **p0** (bits 31—16)	16	R/W	data	signed	DAU
p0l	Low half of **p0** (bits 15—0)	16	R/W	data	signed	DAU
p1	Product 1	32	R/W	data	signed	DAU
p1h	High half of **p1** (bits 31—16)	16	R/W	data	signed	DAU
p1l	Low half of **p1** (bits 15—0)	16	R/W	data	signed	DAU
pi	Program interrupt return	20	R/W	address	unsigned	XAAU
pllc	Phase-lock loop control	16	R/W	control	unsigned	Clocks
powerc	Power control	16	R/W	control	unsigned	Clocks
pr	Program return	20	R/W	address	unsigned	XAAU
psw0, psw1	Program status words 0 and 1	16	R/W[††]	c & s	unsigned	DAU
pt0, pt1	Pointers 0 and 1 to X-memory space	20	R/W	address	unsigned	XAAU
ptrap	Program trap return	20	R/W	address	unsigned	XAAU
r0, r1, r2, r3, r4, r5, r6, r7	Pointers 0—7 to Y-memory space	20	R/W	address	unsigned	YAAU
rb0, rb1	Circular buffer pointers 0 and 1 (begin address)	20	R/W	address	unsigned	YAAU
re0, re1	Circular buffer pointers 0 and 1 (end address)	20	R/W	address	unsigned	YAAU
sbit	BIO status/control	16	R/W	c & s	unsigned	BIO
sp	Stack pointer	20	R/W	address	unsigned	YAAU
timer0, timer1	Timer running count 0 and 1 for Timer0 and Timer1	16	R/W	data	unsigned	Timer
timer0c, timer1c	Timer control 0 and 1 for Timer0 and Timer1	16	R/W	control	unsigned	Timer
vbase	Vector base offset	20	R/W	address	unsigned	XAAU
vsw	Viterbi support word	16	R/W	control	unsigned	DAU
x	Multiplier input	32	R/W	data	signed	DAU
xh	High half of **x** (bits 31—16)	16	R/W	data	signed	DAU
xl	Low half of **x** (bits 15—0)	16	R/W	data	signed	DAU
y	Multiplier input	32	R/W	data	signed	DAU
yh	High half of **y** (bits 31—16)	16	R/W	data	signed	DAU
yl	Low half of **y** (bits 15—0)	16	R/W	data	signed	DAU

† R indicates that the register is readable by instructions; W indicates the register is writable by instructions.
‡ c & s means control and status.
§ Signed registers are in two's complement format.
†† Some bits in the **psw0** and **psw1** registers are read only (writes to these bits are ignored).

Software Architecture (continued)

Registers (continued)

Register Overview (continued)

Table 39 lists the DSP16210 ESIO memory-mapped registers. Table 40 lists registers that are accessible only through MIOU commands. Table 41 lists registers that are DMA-accessible through IORAM.

Table 39. ESIO Memory-Mapped Registers

Register Name	Description	Size (bits)	R/W†	Type
ICR	Input control register	16	R/W	control
ICSB⟨0—7⟩	Input channel start bit registers 0 through 7	16	R/W	control
ICSL⟨0—1⟩	Input channel sample length registers 0 and 1	16	R/W	control
ICVV	Input channel valid vector register	16	R/W	control
IDMX⟨0—15⟩	Input demultiplexer registers 0 through 15	16	R	data
OCR	Output control register	16	R/W	control
OCSB⟨0—7⟩	Output channel start bit registers 0 through 7	16	R/W	control
OCSL⟨0—1⟩	Output channel sample length registers 0 and 1	16	R/W	control
OCVV	Output channel valid vector register	16	R/W	control
OMX⟨0—15⟩	Output multiplexer registers 0 through 15	16	W	data

† R indicates that the register is indirectly readable by instructions; W indicates the register is indirectly writable by instructions.

Table 40. MIOU-Accessible Registers

Register Name	Description	Size (bits)	R/W†	Type	Signed‡/ Unsigned
IBAS⟨0—1⟩	MIOU⟨0—1⟩ input base address registers	10	W	control	unsigned
ILEN⟨0—1⟩	MIOU⟨0—1⟩ input length registers	12	W	control	signed
ILIM⟨0—1⟩	MIOU⟨0—1⟩ input limit address registers	10	W	control	unsigned
OBAS⟨0—1⟩	MIOU⟨0—1⟩ output base address registers	10	W	control	unsigned
OLEN⟨0—1⟩	MIOU⟨0—1⟩ output length registers	11	W	control	unsigned
OLIM⟨0—1⟩	MIOU⟨0—1⟩ output limit address registers	10	W	control	unsigned
PHIFC	PHIF16 control register	12	W	control	unsigned
SSIOC	SSIO control register	12	W	control	unsigned

† R indicates that the register is readable by MIOU commands; W indicates the register is writable by MIOU commands.
‡ Signed registers are in two's complement format.

Table 41. DMA-Accessible Registers

Register Name	Description	Accessible Via	Size (bits)	R/W†	Type	Signed/ Unsigned
PDX(in)	PHIF16 input register	IORAM0/MIOU0	16	R	data	unsigned
PDX(out)	PHIF16 output register	IORAM0/MIOU0	16	W	data	unsigned
SSDX(in)	SSIO input register	IORAM1/MIOU1	16	R	data	unsigned
SSDX(out)	SSIO output register	IORAM1/MIOU1	16	W	data	unsigned

† R indicates that the register is readable by DMA; W indicates the register is writable by DMA.

Software Architecture (continued)

Registers (continued)

Register Settings

Tables 42 through 73 describe the programmable registers of the DSP16210 device.

Table 42. alf Register

15	14—10	9	8	7	6	5	4	3	2	1	0
AWAIT	Reserved	JOBF	JIBE	JCONT	LOCK	MBUSY1	MBUSY0	SOMEF	SOMET	ALLF	ALLT

Bit	Field		Value	Description
15	AWAIT†	Await	0	The core is not in low-power standby mode.
			1	Enter low-power standby mode.
14—10	Reserved	—	0	Reserved—write with zero.
9	JOBF	JTAG Output Buffer Full	0	JTAG **jiob** output buffer is empty.
			1	JTAG **jiob** output buffer is full.
8	JIBE	JTAG Input Buffer Empty	0	JTAG **jiob** input buffer is full.
			1	JTAG **jiob** input buffer is empty.
7	JCONT	JTAG Continue	—	JTAG continue flag.
6	LOCK‡	PLL Lock	0	PLL is not phase-locked.
			1	PLL is phase-locked.
5	MBUSY1§	MIOU1 Busy	0	MIOU1 output is complete.
			1	MIOU1 unfinished output is pending.
4	MBUSY0§	MIOU0 Busy	0	MIOU0 output is complete.
			1	MIOU0 unfinished output is pending.
3	SOMEF	BIO Some False (Inverse of ALLT)	0	All tested BIO input pins match the test pattern in **cbit**[7:0], no BIO input pins are tested, or all BIO pins are configured as outputs.
			1	Some tested BIO inputs pins do not match the test pattern in **cbit**[7:0], i.e., no tested BIO pins match the pattern or some (but not all) tested BIO pins match the pattern.
2	SOMET	BIO Some True (Inverse of ALLF)	0	No tested BIO input pins match the test pattern in **cbit**[7:0], no BIO input pins are tested, or all BIO pins are configured as outputs.
			1	Some or all tested BIO input pins match the test pattern in **cbit**[7:0].
1	ALLF	BIO All False (Inverse of SOMET)	0	Some or all tested BIO input pins match the test pattern in **cbit**[7:0].
			1	No tested BIO input pins match the test pattern in **cbit**[7:0], no BIO input pins are tested, or all BIO pins are configured as outputs.
0	ALLT	BIO All True (Inverse of SOMEF)	0	Some tested BIO inputs pins do not match the test pattern in **cbit**[7:0], i.e., no tested BIO pins match the pattern or some (but not all) tested BIO pins match the pattern.
			1	All tested BIO input pins match the test pattern in **cbit**[7:0], no BIO input pins are tested, or all BIO pins are configured as outputs.

† The AWAIT bit is the only bit in **alf** that is cleared on reset.
‡ LOCK is cleared whenever the **pllc** register is written.
§ The MBUSY1 and MBUSY0 flags are read only (writes to these flags are ignored).

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 43. auc0 (Arithmetic Unit Control 0) Register

15—14	13—11	10	9	8	7	6	5—4	3—2	1—0
P1SHFT[1:0]	Reserved	FSAT	SHFT15	RAND	X=Y=	YCLR	ACLR[1:0]	ASAT[1:0]	P0SHFT[1:0]

Bit	Field	Value	Description
15—14	P1SHFT[1:0]	00	**p1** not shifted.
		01	**p1**>>2.
		10	**p1**<<2.
		11	**p1**<<1.
13—11	Reserved	—	Reserved—write with zero.
10	FSAT	0	Disabled when zero.
		1	Enable 32-bit saturation for the following results: the scaled outputs of the **p0** and **p1** registers, the intermediate result of the 3-input ADDER[†], and the results of the ALU/ACS, ADDER/ACS, and BMU.
9	SHFT15	0	**p1**>>15 in F1E operations performs normally.
		1	To support GSM-EFR, **p1**>>15 in F1E operations actually performs (**p1**>>16)<<1 clearing the least significant bit.
8	RAND	0	Enable pseudorandom sequence generator (PSG)[‡].
		1	Reset and disable pseudorandom sequence generator (PSG).
7	X=Y=	0	Normal operation.
		1	Data transfer statements that load the **y** register also load the **x** register with the same value[§].
6	YCLR	0	The DAU clears **yl** if it loads **yh**.
		1	The DAU leaves **yl** unchanged if it loads **yh**.
5	ACLR[1]	0	The DAU clears **a1l** if it loads **a1h**.
		1	The DAU leaves **a1l** unchanged if it loads **a1h**.
4	ACLR[0]	0	The DAU clears **a0l** if it loads **a0h**.
		1	The DAU leaves **a0l** unchanged if it loads **a0h**.
3	ASAT[1]	0	Enable **a1** saturation[††] on 32-bit overflow.
		1	Disable **a1** saturation on 32-bit overflow.
2	ASAT[0]	0	Enable **a0** saturation[††] on 32-bit overflow.
		1	Disable **a0** saturation on 32-bit overflow.
1—0	P0SHFT[1:0]	00	**p0** not shifted.
		01	**p0**>>2.
		10	**p0**<<2.
		11	**p0**<<1.

† Saturation takes effect only if the ADDER has three input operands and there is no ALU/ACS operation in the same instruction.
‡ After re-enabling the PSG by clearing RAND, the program must wait one instruction cycle before testing the heads or tails condition.
§ The following apply:
 ▪ Instructions that explicitly load any part of the **x** register (i.e., **x**, **xh**, or **xl**) take precedence over the X=Y= mode.
 ▪ Instructions that load **yh** (but not **x** or **xh**) load **xh** with the same data. If YCLR is zero, the DAU clears **yl** and **xl**.
 ▪ Instructions that load **yl** load **xl** with the same data and leave **yh** and **xh** unchanged.
†† If enabled, 32-bit saturation of the accumulator value occurs if the DAU stores the value to memory or to a register. Saturation also applies if the DAU stores the low half, high half, or guard bits of the accumulator. There is no change to the contents stored in the accumulator; only the value stored to memory or a register is saturated.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 44. auc1 (Arithmetic Unit Control 1) Register

15	14—12	11—6	5—0
Reserved	XYFBK[2:0]	ACLR[7:2]	ASAT[7:2]

Bit	Field	Value	Description
15	Reserved	—	Reserved—write with zero.
14—12	XYFBK[2:0][†]	000	Normal operation.
		001	Any DAU function result stored into **a6**[31:0] is also stored into **x**.[‡]
		010	Any DAU function result stored into **a6**[31:16] is also stored into **xh**.[‡]
		011	Any DAU function result stored into **a6**[31:16] is also stored into **xh** and any DAU function result stored into **a7**[31:16] is also stored into **xl**.[‡]
		100	Reserved.
		101	Any DAU function result stored into **a6**[31:0] is also stored into **y**.[§]
		110	Any DAU function result stored into **a6**[31:16] is also stored into **yh**.[§††]
		111	Any DAU function result stored into **a6**[31:16] is also stored into **yh** and any DAU function result stored into **a7**[31:16] is also stored into **yl**.[§‡‡]
11	ACLR[7]	0	The DAU clears **a7l** if it loads **a7h**.
		1	The DAU leaves **a7l** unchanged if it loads **a7h**.
10	ACLR[6]	0	The DAU clears **a6l** if it loads **a6h**.
		1	The DAU leaves **a6l** unchanged if it loads **a6h**.
9	ACLR[5]	0	The DAU clears **a5l** if it loads **a5h**.
		1	The DAU leaves **a5l** unchanged if it loads **a5h**.
8	ACLR[4]	0	The DAU clears **a4l** if it loads **a4h**.
		1	The DAU leaves **a4l** unchanged if it loads **a4h**.
7	ACLR[3]	0	The DAU clears **a3l** if it loads **a3h**.
		1	The DAU leaves **a3l** unchanged if it loads **a3h**.
6	ACLR[2]	0	The DAU clears **a2l** if it loads **a2h**.
		1	The DAU leaves **a2l** unchanged if it loads **a2h**.
5	ASAT[7]	0	Enable **a7** saturation[§§] on 32-bit overflow.
		1	Disable **a7** saturation on 32-bit overflow.
4	ASAT[6]	0	Enable **a6** saturation[§§] on 32-bit overflow.
		1	Disable **a6** saturation on 32-bit overflow.
3	ASAT[5]	0	Enable **a5** saturation[§§] on 32-bit overflow.
		1	Disable **a5** saturation on 32-bit overflow.
2	ASAT[4]	0	Enable **a4** saturation[§§] on 32-bit overflow.
		1	Disable **a4** saturation on 32-bit overflow.
1	ASAT[3]	0	Enable **a3** saturation[§§] on 32-bit overflow.
		1	Disable **a3** saturation on 32-bit overflow.
0	ASAT[2]	0	Enable **a2** saturation[§§] on 32-bit overflow.
		1	Disable **a2** saturation on 32-bit overflow.

† If the application enables any of the XYFBK modes, i.e., XYFBK[2:0]≠000, the following apply:
  ■ Only if the DAU writes its result to **a6** or **a7** (e.g., **a6**=**a3**+**p0**) will the result be written to **x** or **y**. Data transfers or data move operations (e.g., **a6**=*r2) leave the **x** or **y** register unchanged regardless of the state of the XYFBK[2:0] field setting.
  ■ If the instruction itself loads the same portion of the **x** or **y** register that the XYFBK[2:0] field specifies, the instruction load takes precedence.
‡ If the application enables the X=Y= mode (**auc0**[7]=1), the XYFBK mode takes precedence.
§ If the application enables the X=Y= mode (**auc0**[7]=1), the DAU also writes the **y** register value into the **x**, **xh**, or **xl** register as appropriate.
†† If the application enables the YCLR mode (**auc0**[6]=0), the DAU clears **yl**.
‡‡ If the application enables the YCLR mode (**auc0**[6]=0) and the instruction contains a result written to **a6** and the operation writes no result to **a7**, the DAU clears **yl**. If the application enables the YCLR mode and the instruction writes a result to **a7**, the XYFBK mode takes precedence and the DAU does not clear **yl**.
§§ If saturation is enabled and any portion of an accumulator is stored to memory or a register, the DAU saturates the entire accumulator value and stores the appropriate portion. The DAU does not change the contents of the accumulator.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 45. cbit (BIO Control) Register

15—8		7—0	
MODE/MASK[7:0]		DATA/PAT[7:0]	

DIREC[n][†]	MODE/ MASK[n][†]	DATA/ PAT[n][†]	Action
1 (Output)	0	0	Clear
1 (Output)	0	1	Set
1 (Output)	1	0	No Change to register
1 (Output)	1	1	Toggle
0 (Input)	0	0	No Test
0 (Input)	0	1	No Test
0 (Input)	1	0	Test for Zero
0 (Input)	1	1	Test for One

† $0 \le n \le 7$. DIREC[n] is a field in the **sbit** register.

Table 46. cstate (Cache State) Register

15	14	13	12—10	9—5	4—0
SU	EX	LD	Reserved	PTR[4:0]	N[4:0]

Bit	Field	Value	Description
15	SU	0	Normal operation—core not in interrupt/trap handler.
		1	Core suspends cache operation for interrupt or trap.
14	EX	0	Core currently loading cache or not in cache loop.
		1	Core currently executing from within the cache (iteration 2 and up).
13	LD	0	Core currently executing from within the cache or not in cache loop.
		1	Core currently loading cache (iteration 1).
12—10	Reserved	—	Reserved—write with zero.
9—5	PTR[4:0]	0—30	Pointer to current instruction in cache to load or execute.
4—0	N[4:0]	0[†]—31	Number of instructions in the cache loop to load/save/restore.

† N[4:0] is cleared on reset. After execution of the first **do K** or **do cloop** instruction, N[4:0] contains a nonzero value.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 47. ICR (ESIO Input Control) Register

Note: This register is not directly program-accessible (memory-mapped to address 0xE001A).

15—12	11—10	9	8	7	6	5	4	3	2	1—0
Res	IFRMSZ[1:0]	ITMODE	IMODE	ISIZE	ISDLY	ISLEV	IRESET	ILEV	ICA	IFIR[1:0]

Bit	Field	Value	Description
15—12	Res	—	Reserved—write with zero.
11—10	IFRMSZ[1:0] (frame mode only)	00	256-bit frame size (default).
		01	192-bit frame size.
		10	128-bit frame size.
		11	64-bit frame size.
9	ITMODE (frame mode only)	0	No override of **ICSL⟨0—1⟩** register transfer rate control.
		1	Override **ICSL⟨0—1⟩** register transfer rate control by transferring all channel input shift registers to the associated **IDMX⟨0—15⟩** registers at the IFIR frequency.
8	IMODE	0	Select frame mode.
		1	Select simple mode.
7	ISIZE	0	16-bit mode: **IDMX⟨0—15⟩** registers start shift-in at bit 15.
		1	8-bit mode: **IDMX⟨0—15⟩** registers start shift-in at bit 7.
6	ISDLY	0	No action.
		1	Synchronize internal input frame sync (IFS) with the ESIO input bit clock (EIBC) pin.
5	ISLEV	0	Do not invert the ESIO input frame sync (EIFS) pin to produce the internal input frame sync (IFS) signal.
		1	Invert the EIFS pin to produce the internal IFS signal.
4	IRESET	0	No action. This bit always reads as zero.
		1	Reset the ESIO input section—the ESIO automatically clears this bit one CLK cycle after performing the reset.
3	ILEV	0	Do not invert the ESIO input bit clock (EIBC) pin to produce the internal input bit clock (IBC) signal.
		1	Invert the EIBC pin to produce the internal IBC signal.
2	ICA	0	Disable ESIO input section—no input processing.
		1	Enable ESIO input section—input processing.
1—0	IFIR[1:0] (frame mode only)	00	Input frame interrupt rate is every two complete input frames.
		01	Input frame interrupt rate is every four complete input frames.
		10	Input frame interrupt rate is every eight complete input frames.
		11	Input frame interrupt rate is every sixteen complete input frames.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 48. ICSB⟨0—7⟩ (ESIO Input Channel Start Bit) Registers

Note: These registers are not directly program-accessible (memory-mapped to addresses 0xE0010—0xE0017).

	15—8	7—0
ICSB0	Channel 1	Channel 0
ICSB1	Channel 3	Channel 2
ICSB2	Channel 5	Channel 4
ICSB3	Channel 7	Channel 6
ICSB4	Channel 9	Channel 8
ICSB5	Channel 11	Channel 10
ICSB6	Channel 13	Channel 12
ICSB7	Channel 15	Channel 14

Field	Value	Description
Channel 0 to Channel 15	0x00 to 0xFF	Start bit position for corresponding logical input channel. Ranges from 0 to 255.

Table 49. ICSL⟨0—1⟩ (ESIO Input Channel Sample Length) Registers

Note: These registers are used only in frame mode (**ICR**[IMODE] = 0) and are not directly program-accessible (memory-mapped to addresses 0xE0018—0xE0019).

	15—14	13—12	11—10	9—8	7—6	5—4	3—2	1—0
ICSL0	Channel 7	Channel 6	Channel 5	Channel 4	Channel 3	Channel 2	Channel 1	Channel 0
ICSL1	Channel 15	Channel 14	Channel 13	Channel 12	Channel 11	Channel 10	Channel 9	Channel 8

Field	Value	Description
Channel 0 to Channel 15	00	Input sample length is 1 bit (serial-to-parallel transfer rate is every 16 frames).
	01	Input sample length is 2 bits (serial-to-parallel transfer rate is every 8 frames).
	10	Input sample length is 4 bits (serial-to-parallel transfer rate is every 4 frames).
	11	Input sample length is 8 bits (serial-to-parallel transfer rate is every 2 frames).

Table 50. ICVV (ESIO Input Channel Valid Vector) Register

Note: This register is not directly program-accessible (memory-mapped to address 0xE001B). For simple mode, enable only logical channel 0, i.e., set **ICVV** to 0x0001. For frame mode, the bits in **ICVV** must be packed, i.e., channels must be allocated from 0 to 15 with no holes between valid channels. For example, if **ICVV** contains 0x00FF, then logical channels 0—7 are enabled and demultiplexed. A value of 0x08FF for **ICVV** is invalid because the channels are not packed.

15	14	13	12	11	10	9	8
Channel 15	Channel 14	Channel 13	Channel 12	Channel 11	Channel 10	Channel 9	Channel 8

7	6	5	4	3	2	1	0
Channel 7	Channel 6	Channel 5	Channel 4	Channel 3	Channel 2	Channel 1	Channel 0

Field	Value	Description
Channel 0 to Channel 15	0	Disable the corresponding logical input channel, i.e., do not demultiplex the input data stream for this logical channel.
	1	Enable the corresponding logical input channel, i.e., demultiplex the input data stream for this logical channel.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 51. ID (JTAG Identification) Register

Note: This register is not directly program-accessible. It is accessible via the JTAG port.

31—28	27—19	18—12	11—0
DEVICE OPTIONS	ROMCODE	PART ID	LUCENT ID

Bit	Field	Value	Features
31—28	DEVICE OPTIONS	0x3	Device options.
27—19	ROMCODE	0x190	ROMCODE of device.
18—12	PART ID	0x10	DSP16210.
11—0	LUCENT ID	0x03B	Lucent identification.

Table 52. inc⟨0—1⟩ (Interrupt Control) Registers

	19—18	17—16	15—14	13—12	11—10
inc0	TIME0[1:0]	INT3[1:0]	INT2[1:0]	INT1[1:0]	INT0[1:0]
inc1	Reserved—write with zero				EOBE[1:0]
	9—8	**7—6**	**5—4**	**3—2**	**1—0**
inc0	MOBE1[1:0]	MIBF1[1:0]	MOBE0[1:0]	MIBF0[1:0]	Reserved
inc1	EIBF[1:0]	ECOL[1:0]	EOFE[1:0]	EIFE[1:0]	TIME1[1:0]

Field†	Value	Description
TIME0[1:0] INT3[1:0] INT2[1:0] INT1[1:0]	00	Disable the selected interrupt (no priority).
INT0[1:0] MOBE1[1:0] MIBF1[1:0] MOBE0[1:0]	01	Enable the selected interrupt at priority 1 (lowest).
MIBF0[1:0] EOBE[1:0] EIBF[1:0] ECOL[1:0]	10	Enable the selected interrupt at priority 2.
EOFE[1:0] EIFE[1:0] TIME1[1:0]	11	Enable the selected interrupt at priority 3 (highest).

† Reset clears all fields to disable all interrupts.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 53. ins (Interrupt Status) Register

Note: An instruction clears an interrupt and resets its source to the not pending state by writing a one to the corresponding bit in the **ins** register. Writing a zero to any bit leaves the bit unchanged.

19—16				15	14	13	12	11	10
Reserved				EOBE	EIBF	ECOL	EOFE	EIFE	TIME1
9	8	7	6	5	4	3	2	1	0
TIME0	INT3	INT2	INT1	INT0	MOBE1	MIBF1	MOBE0	MIBF0	Reserved

Bit	Field†	Value	Description
19—16	Reserved	—	Reserved—write with zero.
15—0	EOBE EIBF ECOL EOFE EIFE TIME1 TIME0 INT3 INT2 INT1 INT0 MOBE1 MIBF1 MOBE0 MIBF0	0	Read—corresponding interrupt not pending. Write—no effect.
		1	Read—corresponding interrupt is pending. Write—clears bit and changes corresponding interrupt status to not pending‡.

† The core clears an interrupt's **ins** bit if it services that interrupt. For interrupt polling, an instruction can explicitly clear an interrupt's **ins** bit by writing a 1 to that bit and a 0 to all other **ins** bits. Writing a 0 to any **ins** bit leaves the bit unchanged.

‡ To clear an interrupt's status, an application writes a 1 to the corresponding bit.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 54. ioc (I/O Configuration) Register

15—11	10	9	8	7—5	4	3	2	1	0
Reserved	WDDLY	SIOLB	EBIO	CKOSEL[2:0]	WEROM	RWNADV	DENB2	DENB1	DENB0

Bit	Field	Value	Description
15—11	Reserved	—	Reserved—write with zero.
10	WDDLY (Write Data Delay)	0	Drive write data onto DB[15:0] approximately one half-cycle of CKO† after RWN goes low.
		1	Drive write data onto DB[15:0] approximately one cycle of CKO† after RWN goes low.
9	SIOLB (SIO Loopback)	0	SSIO: Deselect loopback.
		1	SSIO: Select loopback, i.e., loop back DO to DI.
8	EBIO (Enable BIO)	0	Pin Multiplexing: Select VEC[3:0] for the VEC[3:0]/IOBIT[7:4]‡ pins.
		1	Pin Multiplexing: Select the high half of BIO, IOBIT[7:4], for the VEC[3:0]/IOBIT[7:4]‡ pins.
7—5	CKOSEL[2:0] (Selection Control for CKO output pin)	000	CLK: Internal free-running clock.
		001	CLKE: Internal free-running clock suspended (held high) during low-power standby mode (AWAIT bit of **alf** register is set).
		010	ZERO: Held low.
		011	Reserved.
		100	CKI: Output of CKI clock input buffer.
		101	ZERO: Held low.
		110	ONE: Held high.
		111	ONE: Held high.
4	WEROM (Write EROM)	0	Selects YMAP0. This allows for external ERAMHI and ERAMLO requests.
		1	Selects YMAP1. Forces all ERAM requests to access EROM instead. If WEROM is set, the DENB1 field (**ioc** bit 1) and the RDYEN1 and YATIM[3:0] fields (**mwait** bits 13 and 7—4) control Y-side accesses to EROM.
3	RWNADV (RWN Pin Advance)	0	Delay leading edge of RWN.
		1	Do not delay RWN.
2	DENB2 (Delay Enable)	0	Do not delay IO enable.
		1	Delay leading edge of IO enable by one half-cycle of CKO†.
1	DENB1 (Delay Enable)	0	Do not delay ERAM, ERAMHI, and ERAMLO enables.
		1	Delay leading edge of ERAM, ERAMHI, and ERAMLO enables by one half-cycle of CKO†.
0	DENB0 (Delay Enable)	0	Do not delay EROM enable.
		1	Delay leading edge of EROM enable by one half-cycle of CKO†.

† Assuming that the CKO pin is programmed as the internal clock CLK, i.e., CKOSEL[2:0] = 000.
‡ VEC0 corresponds to IOBIT7, VEC1 corresponds to IOBIT6, VEC2 corresponds to IOBIT5, and VEC3 corresponds to IOBIT4.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 55. mcmd⟨0—1⟩ (MIOU⟨0—1⟩ Command) Registers

15—12	11—0
Opcode[3:0]	Parameter[11:0]

Opcode[3:0]	Parameter[11:0]		Command	Command Mnemonic	Action
0x0	10-bit IORAM input buffer base address.	0x*NNN*†	Load **IBAS⟨0,1⟩**	IBAS⟨0,1⟩_LD	**IBAS⟨0,1⟩** ← 0x*NNN*
0x1	10-bit IORAM input buffer limit address.	0x*NNN*†	Load **ILIM⟨0,1⟩**	ILIM⟨0,1⟩_LD	**ILIM⟨0,1⟩** ← 0x*NNN*
0x2	10-bit IORAM output buffer base address.	0x*NNN*†	Load **OBAS⟨0,1⟩**	OBAS⟨0,1⟩_LD	**OBAS⟨0,1⟩** ← 0x*NNN*
0x3	10-bit IORAM output buffer limit address.	0x*NNN*†	Load **OLIM⟨0,1⟩**	OLIM⟨0,1⟩_LD	**OLIM⟨0,1⟩** ← 0x*NNN*
0x4	11-bit unsigned input length update amount.	0x*NNN*‡	Update **ILEN⟨0,1⟩**	ILEN⟨0,1⟩_UP	**ILEN⟨0,1⟩** ← **ILEN⟨0,1⟩** + 0x*NNN* Activate§ peripheral service in MIOU⟨0,1⟩.
0x5	11-bit unsigned output length update amount.	0x*NNN*‡	Update **OLEN⟨0,1⟩**	OLEN⟨0,1⟩_UP	**OLEN⟨0,1⟩** ← **OLEN⟨0,1⟩** + 0x*NNN*
0x6	Must be zero.	0x000	Reset MIOU⟨0,1⟩	RESET⟨0,1⟩	Initialize MIOU⟨0,1⟩ control state and deactivate†† MIOU⟨0,1⟩ peripheral service. See Table 19 for the effect of reset on MIOU⟨0,1⟩ interrupts and registers.
0x7	12-bit value for peripheral control register (**PHIFC** or **SSIOC**‡‡).	0x*NNN*	Load Peripheral Control	PCTL⟨0,1⟩_LD	**PHIFC** ← 0x*NNN* (for MIOU0) or **SSIOC** ← 0x*NNN* (for MIOU1)
0x8	Must be zero.	0x000	Input Disable	INPT⟨0,1⟩_DS	Disable MIOU⟨0,1⟩ input processing. (Input processing is re-enabled by executing a subsequent ILEN⟨0,1⟩_UP command.)
0x9—0xF	Reserved.				

† 0x*NNN* is a 12-bit number for which the ten least significant bits (bits [9:0]) are an IORAM⟨0,1⟩ address and the two most significant bits (bits [11:10]) must be 0.
‡ 0x*NNN* is a 12-bit unsigned number for which the most significant bit (bit 11) must be 0.
§ Or reactivate peripheral service in MIOU⟨0,1⟩ if it has been deactivated by a prior RESET⟨0,1⟩ command.
††Subsequent execution of an ILEN_UP⟨0,1⟩ command reactivates MIOU⟨0,1⟩ peripheral service.
‡‡See Table 63 on page 104 and Table 70 on page 110.

Table 56. miwp⟨0—1⟩ (MIOU⟨0—1⟩ IORAM Input Write Pointer) Registers

15—10	9—0
Reserved—write with zero	Write Pointer Address

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 57. morp⟨0—1⟩ (MIOU⟨0—1⟩ IORAM Output Read Pointer) Registers

15—10	9—0
Reserved—write with zero	Read Pointer Address

Table 58. mwait (EMI Configuration) Register

15	14—12	11—8	7—4	3—0
Reserved	RDYEN[2:0]	IATIM[3:0]	YATIM[3:0]	XATIM[3:0]

Bit	Field	Value	Description
15	Reserved	—	Reserved—write with zero.
14	RDYEN2[†]	0	Ignore READY pin for EMI accesses to IO space.
		1	Permit READY pin to extend IO space accesses.
13	RDYEN1[†]	0	Ignore READY pin for EMI accesses to ERAMLO and ERAMHI space.
		1	Permit READY pin to extend ERAMLO and ERAMHI space accesses.
12	RDYEN0[†]	0	Ignore READY pin for EMI accesses to EROM space.
		1	Permit READY pin to extend EROM space accesses.
11—8	IATIM[3:0][‡]	XXXX	Number of DSP clock cycles (CLK) the enable for IO space is asserted.
7—4	YATIM[3:0][‡]	XXXX	Number of DSP clock cycles (CLK) the enable for ERAMLO or ERAMHI space is asserted.
3—0	XATIM[3:0][‡]	XXXX	Number of DSP clock cycles (CLK) the enable for EROM space is asserted.

† These fields are cleared on reset.

‡ These fields cannot be programmed to 0000. If the program writes 0000 to any of these fields, the EMI hardware writes 0001 into the field. The number of cycles per access (wait-state) is the enable assertion time plus one. These fields are set on reset (**mwait** resets to 0x0FFF).

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 59. OCR (ESIO Output Control) Register

Note: This register is not directly program-accessible (memory-mapped to address 0xE003A).

15—14	13—12	11	10	9	8	7	6	5	4	3	2	1—0
Res	OFRMSZ[1:0]	OTMODE	OMODE	OSIZE	OSLEV	CRESET	EDOEO	EDOMD	ORESET	OLEV	OCA	OFIR[1:0]

Bit	Field	Value	Description
15—14	Res	—	Reserved—write with zero.
13—12	OFRMSZ[1:0] (frame mode only)	00	256-bit frame size (default).
		01	192-bit frame size.
		10	128-bit frame size.
		11	64-bit frame size.
11	OTMODE (frame mode only)	0	No override of **OCSL⟨0—1⟩** transfer rate control.
		1	**OCSL⟨0—1⟩** transfer rate control by transferring all **OMX⟨0—15⟩** registers to the associated serial output registers at the OFIR frequency.
10	OMODE	0	Frame mode.
		1	Simple mode.
9	OSIZE	0	When OMODE = 1: 16-bit simple mode. When OMODE = 0: 8-bit frame mode.
		1	When OMODE = 1: 8-bit simple mode. When OMODE = 0: 8-bit frame mode.
8	OSLEV	0	Do not invert the ESIO output frame sync (EOFS) pin to produce the internal output frame sync (OFS) signal.
		1	Invert the EOFS pin to produce the internal OFS signal.
7	CRESET (frame mode only)	0	No action.
		1	Output collision error reset: Clear ECOL interrupt.
6	EDOEO	0	EDO is in high-impedance state.
		1	EDO is enabled.
5	EDOMD	0	EDO is a 3-state driver.
		1	EDO is an open-drain driver.
4	ORESET	0	No action. This bit always reads as zero.
		1	Reset the ESIO output section—the ESIO automatically clears this bit one CLK cycle after performing the reset.
3	OLEV	0	Do not invert the ESIO output bit clock (EOBC) pin to produce the internal output bit clock (OBC) signal.
		1	Invert the EOBC pin to produce the OBC signal.
2	OCA	0	Disable ESIO output section—no output processing.
		1	Enable ESIO output section—output processing.
1—0	OFIR[1:0] (frame mode only)	00	Output frame interrupt rate is every two complete output frames.
		01	Output frame interrupt rate is every four complete output frames.
		10	Output frame interrupt rate is every eight complete output frames.
		11	Output frame interrupt rate is every sixteen complete output frames.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 60. OCSB⟨0—7⟩ (ESIO Output Channel Start Bit) Registers

Note: These registers are not directly program-accessible (memory-mapped at address 0xE0030 to 0xE0037).

	15—8	7—0
OCSB0	Channel 1	Channel 0
OCSB1	Channel 3	Channel 2
OCSB2	Channel 5	Channel 4
OCSB3	Channel 7	Channel 6
OCSB4	Channel 9	Channel 8
OCSB5	Channel 11	Channel 10
OCSB6	Channel 13	Channel 12
OCSB7	Channel 15	Channel 14

Field	Value	Description
Channel 0 to Channel 15	0x00 to 0xFF	Start bit position for corresponding logical output channel. Ranges from 0 to 255.

Table 61. OCSL⟨0—1⟩ (ESIO Output Channel Sample Length) Registers

Note: These registers are used only in frame mode (**OCR**[OMODE] = 0) and are not directly program-accessible (memory-mapped to addresses 0xE0038 and 0xE0039).

	15—14	13—12	11—10	9—8	7—6	5—4	3—2	1—0
OCSL0	Channel 7	Channel 6	Channel 5	Channel 4	Channel 3	Channel 2	Channel 1	Channel 0
OCSL1	Channel 15	Channel 14	Channel 13	Channel 12	Channel 11	Channel 10	Channel 9	Channel 8

Field	Value	Description
Channel 0 to Channel 15	00	Output sample length is 1 bit (parallel-to-serial transfer rate is every 16 frames).
	01	Output sample length is 2 bits (parallel-to-serial transfer rate is every 8 frames).
	10	Output sample length is 4 bits (parallel-to-serial transfer rate is every 4 frames).
	11	Output sample length is 8 bits (parallel-to-serial transfer rate is every 2 frames).

Table 62. OCVV (ESIO Output Channel Valid Vector) Register

Note: This register is not directly program-accessible (memory-mapped to address 0xE003B). For simple mode, enable only logical channel 0, i.e., set **OCVV** to 0x0001. For frame mode, the bits in **OCVV** must be packed, i.e., channels must be allocated from 0 to 15 with no holes between valid channels. For example, if **OCVV** contains 0x00FF, then logical channels 0—7 are enabled and demultiplexed. A value of 0x08FF for **OCVV** is invalid because the channels are not packed.

15	14	13	12	11	10	9	8
Channel 15	Channel 14	Channel 13	Channel 12	Channel 11	Channel 10	Channel 9	Channel 8

7	6	5	4	3	2	1	0
Channel 7	Channel 6	Channel 5	Channel 4	Channel 3	Channel 2	Channel 1	Channel 0

Field	Value	Description
Channel 0 to Channel 15	0	Disable the corresponding logical output channel, i.e., do not multiplex the output data stream for this logical channel.
	1	Enable the corresponding logical output channel, i.e., multiplex the output data stream for this logical channel.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 63. PHIFC (PHIF16 Control) Register

Note: This register is not directly program-accessible. It must be programmed through MIOU0 by writing **mcmd0**.

11—8	7	6	5	4	3	2	1	0
Reserved	PCFIG	PSOBEF	PFLAGSEL	PFLAG	PBSELF	PSTRB	PSTROBE	PMODE

Bit	Field	Value	Description
11—8	Reserved	—	Reserved—write with zero.
7	PCFIG	0	8-bit external bus configuration. PB[15:8] are 3-stated.
		1	16-bit external bus configuration.
6	PSOBEF	0	POBE flag as read through PSTAT register is active-high.
		1	POBE flag as read through PSTAT register is active-low.
5	PFLAGSEL	0	The state of the PIBF pin is same as that of the PIBF flag.
		1	The state of the PIBF pin is the PIBF flag logically ORed with the POBE flag. (The state of the POBE pin is unaffected and is the same as that of the POBE flag.)
4	PFLAG	0	PIBF and POBE pins active-high.
		1	PIBF and POBE pins active-low.
3	PBSELF	0	If PMODE = 0, PBSEL pin = 0 –> PDX0 low byte. If PMODE = 1, PBSEL pin = 0 –> PDX0 low byte. If PMODE = 1, PBSEL pin = 1 –> PDX0 high byte.
		1	If PMODE = 0, PBSEL pin = 1 –> PDX0 low byte. If PMODE = 1, PBSEL pin = 0 –> PDX0 high byte. If PMODE = 1, PBSEL pin = 1 –> PDX0 low byte.
2	PSTRB	0	If PSTROBE = 1, PODS pin (PDS) active-low.
		1	If PSTROBE = 1, PODS pin (PDS) active-high.
1	PSTROBE	0	*Intel* protocol: PIDS and PODS data strobes.
		1	*Motorola* protocol: PRWN and PDS data strobes.
0	PMODE	0	If 8-bit external bus configuration, 8-bit logical data transfers. If 16-bit external bus configuration, preserve high and low byte positions.
		1	If 8-bit external bus configuration, 16-bit logical data transfers. If 16-bit external bus configuration, swap high and low byte positions.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 64. pllc (Phase-Lock Loop) Register

15	14	13	12	11—8	7—5	4—0
PLLEN	PLLSEL	ICP	Reserved	LF[3:0]	Nbits[2:0]	Mbits[4:0]

Bit	Field	Value	Description		
15	PLLEN	0	PLL disabled (powered down).		
		1	PLL enabled (powered up).		
14	PLLSEL	0	DSP core clock (f_{CLK}) taken directly from CKI pin.		
		1	DSP core clock (f_{CLK}) taken from PLL.		
13	ICP	1	Charge pump current selection—set to one for proper operation.		
12	Reserved	0	Reserved—write with zero.		
11—8	LF[3:0]	—	Loop filter setting (see table below.		
7—5	Nbits[2:0]	0—7	Encodes N, $1 \leq N \leq 8$, where N = Nbits[2:0] + 2, unless Nbits = 111, then N = 1.	$f_{PLL} = f_{CKI}$ x M/2N, where: ■ f_{PLL} is the PLL output frequency.	
4—0	Mbits[4:0]	0—22	Encodes M, $2 \leq M \leq 24$, where M = Mbits[4:0] + 2.	■ f_{CKI} is the input clock frequency applied to the CKI pin. ■ Program M/N $\geq$ 2 ($f_{PLL} \geq f_{CKI}$)	

PLL Loop Filter Settings and Lock-In Time

M	pllc[11:8] (LF[3:0])	Typical Lock-In Time (µs)[†]
23—24	1011	30
21—22	1010	30
19—20	1001	30
16—18	1000	30
12—15	0111	30
8—11	0110	30
2—7	0100	30

† Lock-in time is the time following assertion of the PLLEN bit of the **pllc** register during which the PLL output clock is unstable. The DSP must operate from the CKI input clock or from the slow ring oscillator while the PLL is locking. The DSP16210 signals completion of the lock-in interval by setting the LOCK flag.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 65. powerc (Power Control) Register

15—13	12	11	10	9	8	7	6—5	4	3	2	1	0
Res	SSIO	PHIF16	SLOWCLK	NOCK	INT1EN	INT0EN	Res	ESIO	MIOU1	MIOU0	TIMER1	TIMER0

Bit	Field	Value	Description
15—13	Res	—	Reserved—write with zero.
12	SSIO	0	Power up the SSIO (enable the SSIO clock).
		1	Power down SSIO (disable the SSIO clock).
11	PHIF16	0	Power up the PHIF16 (enable the PHIF16 clock).
		1	Power down PHIF16 (disable the PHIF16 clock).
10	SLOWCLK	0	Power down the ring oscillator and deselect it as the internal source clock.
		1	Power up the ring oscillator and select it as the internal source clock. (Overridden by PLLSEL field (**pllc**[14])— PLLSEL = 1, then the PLL is selected as the clock source.)
9	NOCK	0	Enable internal clock operation (CLK).
		1	Disable internal clock operation (CLK), suspending all core, peripheral, and I/O activity until one of the following occurs: ■ INT0 pin is asserted and INT0EN is set. ■ INT1 pin is asserted and INT1EN is set. ■ Device reset.
8	INT1EN	0	Asserting the INT1 pin does not clear the NOCK bit.
		1	Asserting the INT1 pin clears the NOCK bit.
7	INT0EN	0	Asserting the INT0 pin does not clear the NOCK bit.
		1	Asserting the INT0 pin clears the NOCK bit.
6—5	Res	—	Reserved—write with zero.
4	ESIO	0	Power up the ESIO (enable the ESIO clock).
		1	Power down ESIO (disable the ESIO clock).
3	MIOU1	0	Power up the MIOU1 (enable the MIOU1 clock).
		1	Power down MIOU1 (disable the MIOU1 clock).
2	MIOU0	0	Power up the MIOU0 (enable the MIOU0 clock).
		1	Power down MIOU0 (disable the MIOU0 clock).
1	TIMER1	0	Power up the TIMER1 (enable the TIMER1 clock).
		1	Power down TIMER1 (disable the TIMER1 clock).
0	TIMER0	0	Power up the TIMER0 (enable the TIMER0 clock).
		1	Power down TIMER0 (disable the TIMER0 clock).

Table 66. PSTAT (PHIF16 Status) Register

Note: This register is not directly program-accessible. It is accessible via the PHIF16 pins.

7—2	1	0
Reserved	PIBF	POBE

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 67. psw0 (Processor Status Word 0) Register

15	14	13	12	11	10	9	8—5	4	3—0
LMI	LEQ	LLV	LMV	SLLV	SLMV	a1V	a1[35:32]	a0V	a0[35:32]

Bit	Field	Description
15	LMI	Most recent DAU result[†] is negative when set.
14	LEQ	Most recent DAU result[†] is zero (equal) when set.
13	LLV	Most recent DAU operation[†] resulted in logical overflow.[‡]
12	LMV	Most recent DAU operation[†] resulted in mathematical overflow.[§]
11[††]	SLLV	Sticky version of LLV that remains active once set by a DAU operation until explicitly cleared by a write to **psw0**.
10[††]	SLMV	Sticky version of LMV that remains active once set by a DAU operation until explicitly cleared by a write to **psw0**.
9	a1V	a1V is set if an operation results in mathematical overflow, the result is written to **a1**, and FSAT=0.
8—5	a1[35:32]	The four lower guard bits of **a1**.[‡‡]
4	a0V	a0V is set if an operation results in mathematical overflow, the result is written to **a0**, and FSAT=0.
3—0	a0[35:32]	The four lower guard bits of **a0**.[‡‡]

† ALU/ACS result if the DAU operation uses the ALU/ACS; otherwise, ADDER or BMU result, whichever applies.
‡ The ALU or ADDER cannot represent the result in 40 bits or the BMU control operand is out of range.
§ The ALU/ACS, ADDER, or BMU cannot represent the result in 32 bits. For the BMU, other conditions can also cause mathematical overflow.
†† Cleared on reset.
‡‡ Required for compatibility with DSP16XX family.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 68. psw1 (Processor Status Word 1) Register

15	14	13—12	11—10	9—7	6	5—0
Reserved	IEN	IPLc[1:0]	IPLp[1:0]	Reserved	EPAR	a[7:2]V

Bit	Field	Value	Description
15	Reserved	—	Reserved—write with zero.
14†	IEN‡	0	Interrupts are globally disabled§.
		1	Interrupts are globally enabled.
13—12†	IPLc[1:0]	00	Current interrupt priority level is 0; core handles pending interrupts of priority 1, 2, or 3.
		01	Current interrupt priority level is 1; core handles pending interrupts of priority 2 or 3.
		10	Current interrupt priority level is 2; core handles pending interrupts of priority 3 only.
		11	Current interrupt priority level is 3; core does not handle any pending interrupts§.
11—10	IPLp[1:0]	00	Previous interrupt priority level†† was 0.
		01	Previous interrupt priority level†† was 1.
		10	Previous interrupt priority level†† was 2.
		11	Previous interrupt priority level†† was 3.
9—7	Reserved	—	Reserved—write with zero.
6	EPAR	0	Most recent BMU or special function shift result has odd parity.
		1	Most recent BMU or special function shift result has even parity.
5	a7V	1	a7V is set if an operation results in mathematical overflow, the result is written to **a7**, and FSAT=0.
4	a6V	1	a6V is set if an operation results in mathematical overflow, the result is written to **a6**, and FSAT=0.
3	a5V	1	a5V is set if an operation results in mathematical overflow, the result is written to **a5**, and FSAT=0.
2	a4V	1	a4V is set if an operation results in mathematical overflow, the result is written to **a4**, and FSAT=0.
1	a3V	1	a3V is set if an operation results in mathematical overflow, the result is written to **a3**, and FSAT=0.
0	a2V	1	a2V is set if an operation results in mathematical overflow, the result is written to **a2**, and FSAT=0.

† Cleared on reset.
‡ This bit is read only. The programmer clears this bit by executing a **di** instruction and sets it by executing an **ei** or **ireturn** instruction. If the core services an interrupt, it clears this bit.
§ The core handles any pending traps.
†† Previous interrupt priority level is the priority level of the interrupt most recently serviced prior to the current interrupt. This field is used for interrupt nesting.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 69. sbit (BIO Status/Control) Register

15—8			7—0
DIREC[7:0]			VALUE[7:0]

Bit	Field	Value	Description
15—8	DIREC[n]†	1xxxxxxx	IOBIT7 is an output (input when 0).
		x1xxxxxx	IOBIT6 is an output (input when 0).
		xx1xxxxx	IOBIT5 is an output (input when 0).
		xxx1xxxx	IOBIT4 is an output (input when 0).
		xxxx1xxx	IOBIT3 is an output (input when 0).
		xxxxx1xx	IOBIT2 is an output (input when 0).
		xxxxxx1x	IOBIT1 is an output (input when 0).
		xxxxxxx1	IOBIT0 is an output (input when 0).
7—0	VALUE[n]†	Rxxxxxxx	Reads the current value of IOBIT7.
		xRxxxxxx	Reads the current value of IOBIT6.
		xxRxxxxx	Reads the current value of IOBIT5.
		xxxRxxxx	Reads the current value of IOBIT4.
		xxxxRxxx	Reads the current value of IOBIT3.
		xxxxxRxx	Reads the current value of IOBIT2.
		xxxxxxRx	Reads the current value of IOBIT1.
		xxxxxxxR	Reads the current value of IOBIT0.

† $0 \le n \le 7$.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 70. SSIOC (SSIO Control) Register

Note: This register is not directly program-accessible. It must be programmed through MIOU1.

11	10	9	8—7	6	5	4	3	2	1	0
Reserved	DODLY	LD	ACLK[1:0]	MSB	OLD	ILD	OCK	ICK	OLEN	ILEN

Bit	Field	Value	Description
11	Reserved	—	Reserved—write with zero.
10	DODLY	0	DO changes on the rising edge of OCK.
		1	DO changes on the falling edge of OCK. The delay in driving DO increases the hold time on DO by half a cycle of OCK.
9	LD	0	In active mode, ILD and/or OLD = ICK ÷ 16.
		1	In active mode, ILD and/or OLD = OCK ÷ 16.
8—7	ACLK[1:0]	00	Active clock = CLK ÷ 2.
		01	Active clock = CLK ÷ 6.
		10	Active clock = CLK ÷ 8.
		11	Active clock = CLK ÷ 10.
6	MSB	0	LSB first.
		1	MSB first.
5	OLD	0	OLD is an input (passive OLD).
		1	OLD is an output (active OLD).
4	ILD	0	ILD is an input (passive ILD).
		1	ILD is an output (active ILD).
3	OCK	0	OCK is an input (passive OCK).
		1	OCK is an output (active OCK).
2	ICK	0	ICK is an input (passive ICK).
		1	ICK is an output (active ICK).
1	OLEN	0	16-bit output.
		1	8-bit output (passive OLD mode only).
0	ILEN	0	16-bit input.
		1	8-bit input (passive ILD mode only).

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 71. timer⟨0,1⟩ (TIMER⟨0,1⟩ Running Count) Register

	Bits 15—0
Field	Running Count for TIMER⟨0,1⟩[†]
Description	Read **timer⟨0,1⟩** for current output of down counter. Write **timer⟨0,1⟩** to load down counter and period register[‡].
R/W	R/W
Reset Value	0

† To read or write this register, TIMER⟨0,1⟩ must be powered up, i.e., **timer⟨0,1⟩c**[PWR_DWN] and **powerc**[TIMER⟨0,1⟩] must both be cleared.
‡ The period register is used if **timer⟨0,1⟩c**[RELOAD] is set—the timer automatically reloads the down counter from the period register after the counter reaches zero and continues decrementing the counter indefinitely.

Table 72. timer⟨0,1⟩c (TIMER⟨0,1⟩ Control) Register

15—7	6	5	4	3—0
Reserved	PWR_DWN	RELOAD	COUNT	PRESCALE[3:0]

Bit	Field	Value	Description	
15—7	Reserved	0	Reserved—write with zero.	
6	PWR_DWN	0	Power up the timer[†].	
		1	Power down the timer[‡].	
5	RELOAD	0	Stop decrementing the down counter after it reaches zero.	
		1	Automatically reload the down counter from the period register after the counter reaches zero and continue decrementing the counter indefinitely.	
4	COUNT	0	Hold the down counter at its current value, i.e., stop the timer.	
		1	Decrement the down counter, i.e., run the timer.	
3—0	PRESCALE[3:0]	0000	Controls the counter prescaler to determine the frequency of the timer, i.e., the frequency of the clock applied to the timer down counter. This frequency is a ratio of the internal clock frequency f_{CLK}:	$f_{CLK}/2$
		0001		$f_{CLK}/4$
		0010		$f_{CLK}/8$
		0011		$f_{CLK}/16$
		0100		$f_{CLK}/32$
		0101		$f_{CLK}/64$
		0110		$f_{CLK}/128$
		0111		$f_{CLK}/256$
		1000		$f_{CLK}/512$
		1001		$f_{CLK}/1024$
		1010		$f_{CLK}/2048$
		1011		$f_{CLK}/4096$
		1100		$f_{CLK}/8192$
		1101		$f_{CLK}/16384$
		1110		$f_{CLK}/32768$
		1111		$f_{CLK}/65536$

† Except if **powerc**[TIMER⟨0,1⟩] is set.
‡ Except if **powerc**[TIMER⟨0,1⟩] is cleared. If TIMER⟨0,1⟩ is powered down, then **timer⟨0,1⟩** cannot be read or written. While the timer is powered down, the state of the down counter and period register remain unchanged.

Software Architecture (continued)

Registers (continued)

Register Settings (continued)

Table 73. vsw (Viterbi Support Word) Register

15—6			5	4	3	2	1	0
Reserved			VEN	MAX	TB2	Reserved	CFLAG1	CFLAG0

Bit	Field	Value	Description
15—6	Reserved	—	Reserved—write with zero.
5	VEN	0	Disables Viterbi side effects.
		1	Enables Viterbi side effects.
4	MAX	0	The **cmp0( )**, **cmp1( )**, and **cmp2( )** functions select minimum of input operands.
		1	The **cmp0( )**, **cmp1( )**, and **cmp2( )** functions select maximum of input operands.
3	TB2	0	The traceback encoder stuffs one traceback bit into **ar0** for the **cmp1( )** function or stuffs one old traceback bit from **ar0** into **ar1** for the **cmp0( )** function (GSM/IS95-compatible mode).
		1	The traceback encoder stuffs two traceback bits into **ar0** for the **cmp1( )** function or stuffs two old traceback bits from **ar0** into **ar1** for the **cmp0( )** function (IS54/IS136-compatible mode).
2	Reserved	—	Reserved—write with zero.
1	CFLAG1	—	Previous value of CFLAG0. The traceback encoder copies the value of CFLAG0 to CFLAG1 if the DAU executes a **cmp2( )** function and VEN=1.
0	CFLAG0	—	Previous value of CFLAG[†]. The traceback encoder copies the value of CFLAG to CFLAG0 if the DAU executes a **cmp2( )** function and VEN=1.

† For the **cmp2(aSE, aDE)** function, CFLAG=0 if MAX=0 and aSE≥aDE or if MAX=1 and aSE<aDE, and CFLAG=1 if MAX=0 and aSE<aDE or if MAX=1 and aSE≥aDE.

Software Architecture (continued)

Registers (continued)

Reset States

Pin reset occurs if the RSTB pin is asserted (low). (See for more information.) Tables through describe how reset affects the state of the core registers. describes how reset affects the state of the peripheral (off-core) registers. The following bit codes apply:

- Bit code • indicates that this bit is unknown after powerup reset and is unaffected by subsequent pin resets.
- Bit code P indicates the value on the corresponding input pin (applies to **sbit** register and the IOBIT[7:0] pins).

Table 74. Core Register States After Reset—40-bit Registers

Register	Bits 39—0
a0	•••• •••• •••• •••• •••• •••• •••• •••• •••• ••••
a1	•••• •••• •••• •••• •••• •••• •••• •••• •••• ••••
a2	•••• •••• •••• •••• •••• •••• •••• •••• •••• ••••
a3	•••• •••• •••• •••• •••• •••• •••• •••• •••• ••••
a4	•••• •••• •••• •••• •••• •••• •••• •••• •••• ••••
a5	•••• •••• •••• •••• •••• •••• •••• •••• •••• ••••
a6	•••• •••• •••• •••• •••• •••• •••• •••• •••• ••••
a7	•••• •••• •••• •••• •••• •••• •••• •••• •••• ••••

Table 75. Core Register States After Reset—32-bit Registers

Register	Bits 31—0
csave	•••• •••• •••• •••• •••• •••• •••• ••••
p0	•••• •••• •••• •••• •••• •••• •••• ••••
p1	•••• •••• •••• •••• •••• •••• •••• ••••
x	•••• •••• •••• •••• •••• •••• •••• ••••
y	•••• •••• •••• •••• •••• •••• •••• ••••

Table 76. Core Register States After Reset—20-bit Registers

Register	Bits 19—0	Register	Bits 19—0
h	•••• •••• •••• •••• ••••	r1	•••• •••• •••• •••• ••••
i	•••• •••• •••• •••• ••••	r2	•••• •••• •••• •••• ••••
inc0	0000 0000 0000 0000 0000	r3	•••• •••• •••• •••• ••••
inc1	0000 0000 0000 0000 0000	r4	•••• •••• •••• •••• ••••
ins	0000 0000 0000 0000 0000	r5	•••• •••• •••• •••• ••••
j	•••• •••• •••• •••• ••••	r6	•••• •••• •••• •••• ••••
k	•••• •••• •••• •••• ••••	r7	•••• •••• •••• •••• ••••
PC†	XXXX 0000 0000 0000 0000	rb0	0000 0000 0000 0000 0000
pi	•••• •••• •••• •••• ••••	rb1	0000 0000 0000 0000 0000
pr	•••• •••• •••• •••• ••••	re0	0000 0000 0000 0000 0000
pt0	•••• •••• •••• •••• ••••	re1	0000 0000 0000 0000 0000
pt1	•••• •••• •••• •••• ••••	sp	•••• •••• •••• •••• ••••
ptrap	•••• •••• •••• •••• ••••	vbase	0010 0000 0000 0001 0100
r0	•••• •••• •••• •••• ••••		

† **PC** resets to 0x20000 (first address of IROM) if the EXM pin is 0 at the time of reset. It resets to 0x80000 (first address of EROM) if the EXM pin is 1 at the time of reset.

Software Architecture (continued)

Registers (continued)

Reset States (continued)

Table 77. Core Register States After Reset—16-bit Registers

Register	Bits 15—0	Register	Bits 15—0
alf	0000 00•• •••• ••••	c1	•••• •••• •••• ••••
ar0	•••• •••• •••• ••••	c2	•••• •••• •••• ••••
ar1	•••• •••• •••• ••••	cloop	0000 0000 0000 0000
ar2	•••• •••• •••• ••••	cstate	0000 0000 0000 0000
ar3	•••• •••• •••• ••••	psw0	•••• 00•• •••• ••••
auc0	0000 0000 0000 0000	psw1	0000 •••• •••• ••••
auc1	0000 0000 0000 0000	vsw	0000 0000 0000 0000
c0	•••• •••• •••• ••••		

Table 78. Peripheral (Off-Core) Register States After Reset

Register	Size (bits)	Bits 15—0	Register	Size (bits)	Bits 15—0
cbit	16	•••• •••• •••• ••••	OCSL⟨0—1⟩	16	0000 0000 0000 0000
IBAS⟨0—1⟩	10	•• •••• ••••	OCVV	16	0000 0000 0000 0000
ICR	16	0000 0000 0000 0000	OLEN⟨0—1⟩	11	000 0000 0000
ICSB⟨0—7⟩	16	0000 0000 0000 0000	OLIM⟨0—1⟩	10	•• •••• ••••
ICSL⟨0—1⟩	16	0000 0000 0000 0000	OMX⟨0—15⟩	16	•••• •••• •••• ••••
ICVV	16	0000 0000 0000 0000	PDX(in)	16	•••• •••• •••• ••••
IDMX⟨0—15⟩	16	•••• •••• •••• ••••	PDX(out)	16	0000 0000 0000 0000
ILEN⟨0—1⟩	12	1111 1111 1111	PHIFC	12	0000 0000 0000
ILIM⟨0—1⟩	10	•• •••• ••••	pllc	16	0000 0000 0000 0000
ioc	16	0000 0000 0000 0000	powerc	16	0000 0000 0000 0000
mcmd⟨0—1⟩	16	•••• •••• •••• ••••	PSTAT	8	•••• ••01
miwp⟨0—1⟩	16	•••• ••00 0000 0000	sbit†	16	0000 0000 PPPP PPPP
morp⟨0—1⟩	16	•••• ••00 0000 0000	SSDX(in)	16	•••• •••• •••• ••••
mwait‡	16	0000 1111 1111 1111	SSDX(out)	16	•••• •••• •••• ••••
OBAS⟨0—1⟩	10	•• •••• ••••	SSIOC	12	0000 0000 0000
OCR	16	0000 0000 0000 0000	timer⟨0—1⟩	16	0000 0000 0000 0000
OCSB⟨0—7⟩	16	0000 0000 0000 0000	timer⟨0—1⟩c	16	0000 0000 0000 0000
jiob§	32	•••• •••• •••• •••• •••• •••• •••• ••••			

† The value of **sbit**[7:0] is the same as that of the pins IOBIT[7:0].
‡ Unlike the DSP1620, there is no external means (e.g., INT1 and EXM) to initialize **mwait** to any other value.
§ The **jiob** register is the only peripheral register that is 32 bits; therefore, the bit pattern shown is for bits 31—0.

Note: Upon exiting the boot code, the following core registers are not reinitialized to their reset states as defined in the *DSP16000 Digital Signal Processor Core* Information Manual: **inc0**, **rb0**, **re0**, **vbase**, **cloop**, **cstate**. With the exception of the **ioc** register, none of the peripheral registers are reinitialized to their reset states as defined in Table 78. It is recommended that the user code immediately globally disable interrupts by executing a **di** instruction and clear all pending interrupts by clearing **ins** (ins = 0xfffff).

Software Architecture (continued)

### Registers (continued)

RB Field Encoding

Table 79 describes the encoding of the RB field. This information supplements the instruction set encoding information in the *DSP16000 Digital Signal Processor Core Instruction Set* Reference Manual.

Table 79. RB Field

RB[†]	Register	RB[†]	Register	RB[†]	Register	RB[†]	Register
000000	**a0g**	010000	Reserved	100000	**ioc**	110000	**morp1**
000001	**a1g**	010001	**cloop**	100001	**powerc**	110001	Reserved
000010	**a2g**	010010	**cstate**	100010	**pllc**	110010	Reserved
000011	**a3g**	010011	**csave**	100011	Reserved	110011	Reserved
000100	**a4g**	010100	**auc1**	100100	**mwait**	110100	Reserved
000101	**a5g**	010101	**ptrap**	100101	**cbit**	110101	Reserved
000110	**a6g**	010110	**vsw**	100110	**sbit**	110110	Reserved
000111	**a7g**	010111	Reserved	100111	**timer0c**	110111	Reserved
001000	**a0_1h**	011000	**ar0**	101000	**timer0**	111000	Reserved
001001	**inc1**	011001	**ar1**	101001	**timer1c**	111001	Reserved
001010	**a2_3h**	011010	**ar2**	101010	**timer1**	111010	Reserved
001011	**inc0**	011011	**ar3**	101011	**mcmd0**	111011	Reserved
001100	**a4_5h**	011100	**vbase**	101100	**miwp0**	111100	Reserved
001101	**pi**	011101	**ins**	101101	**morp0**	111101	Reserved
001110	**a6_7h**	011110	Reserved	101110	**mcmd1**	111110	Reserved
001111	**psw1**	011111	Reserved	101111	**miwp1**	111111	**jiob**

† RB field specifies one of a secondary set of registers as the destination of a data move. Codes 000000 through 011111 correspond to core registers and codes 100000 through 111111 correspond to off-core (peripheral) registers.

DSP16210 Digital Signal Processor
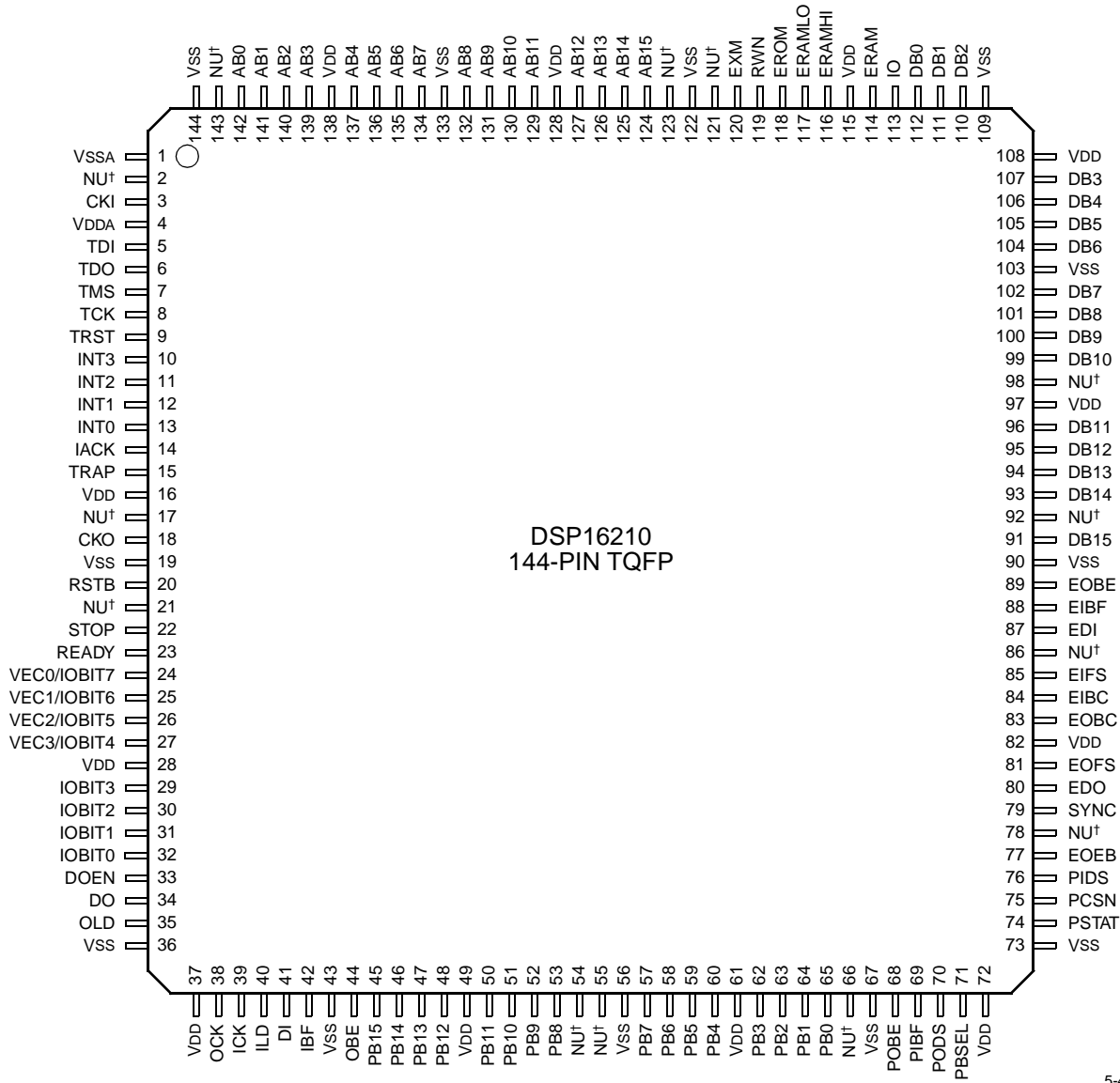
Pin Information

† NU = not usable; no external connections are allowed.

Figure 24. DSP16210 144-Pin TQFP Pin Diagram (Top View)

Lucent Technologies Inc.

Pin Information (continued)

Functional descriptions of TQFP pins 1—144 are found in Signal Descriptions beginning on page 121. Input levels on all I (input) and I/O (input/output) type pins are designed to remain at full CMOS levels when not driven. At full CMOS levels, no significant dc current is drawn. Although input and I/O buffers can be left untied, it is recommended that unused input pins (and I/O pins that are configured as inputs) be tied to VSS or VDD through a 10 kΩ resistor.

Table 80. Pin Descriptions

TQFP Pin	Symbol	Type	Name/Function	Pin State During Reset (RSTB = 0)		Pin State After Reset (RSTB 0 → 1)
91, 93, 94, 95, 96, 99, 100, 101, 102, 104, 105, 106, 107, 110, 111, 112	DB[15:0]	I/O	External Memory Data Bus 15—0	3-state		3-state
113	IO	O	Y-Memory Address Space External I/O Enable	INT0 = 0 (deasserted)	logic high	logic high
				INT0 = 1 (asserted)	3-state	
116	ERAMHI	O	Y-Memory Address Space External RAM High Enable	INT0 = 0 (deasserted)	logic high	logic high
				INT0 = 1 (asserted)	3-state	
117	ERAMLO	O	Y-Memory Address Space External RAM Low Enable	INT0 = 0 (deasserted)	logic high	logic high
				INT0 = 1 (asserted)	3-state	
114	ERAM	O	Y-Memory Address Space External RAM Enable	INT0 = 0 (deasserted)	logic high	logic high
				INT0 = 1 (asserted)	3-state	
118	EROM	O	X-Memory Address Space External ROM Enable	INT0 = 0 (deasserted)	logic high	logic high
				INT0 = 1 (asserted)	3-state	
119	RWN	O	EMI Read/Write Not Indicator	INT0 = 0 (deasserted)	logic high	logic high
				INT0 = 1 (asserted)	3-state	
120	EXM	I	External Memory Boot Select	—		—
23	READY	I	External Memory Access Acknowledge	—		—

† During and after reset, the internal clock is selected as the CKI input pin and the CKO output pin is selected as the internal clock.
‡ This pin his internal pull-up circuitry.
§ 3-states by JTAG control.
†† The **ioc** register (see Table 54 on page 99) is cleared after reset, including its EBIO field that controls the multiplexing of the VEC0/IOBIT7, VEC1/IOBIT6, VDC2/IOBIT5, and VEC3/IOBIT4 pins. After reset, these pins are configured as the VEC[3:0] outputs and are logic high.
‡‡ If unused, this pin must be pulled low through a 10 kΩ resistor to VSS.
§§ 3-states if RSTB = 0 or **PHIFC**[PCFIG] = 0.

Pin Information (continued)

Table 80. Pin Descriptions (continued)

TQFP Pin	Symbol	Type	Name/Function	Pin State During Reset (RSTB = 0)		Pin State After Reset (RSTB 0 → 1)
124, 125, 126, 127, 129, 130, 131, 132, 134, 135, 136, 137, 139, 140, 141, 142	AB[15:0]	O	External Memory Address Bus 15—0	3-state		logic low
10, 11, 12, 13	INT[3:0]	I	External Interrupt Requests	—		—
14	IACK	O	Interrupt Acknowledge	3-state		logic low
22	STOP	I	STOP DSP Clocks (negative assertion)	—		—
15	TRAP	I/O	TRAP/Breakpoint Indication	3-state		configured as input
20	RSTB	I	Device Reset (negative assertion)	—		—
18	CKO	O	Programmable Clock Output	INT0 = 0 (deasserted)	internal clock (CLK = CKI)†	internal clock (CLK = CKI)†
				INT0 = 1 (asserted)	3-state	
8	TCK	I	JTAG Test Clock	—		—
7	TMS	I‡	JTAG Test Mode Select	—		—
6	TDO	O§	JTAG Test Data Output	—		—
5	TDI	I‡	JTAG Test Data Input	—		—
9	TRST	I‡	JTAG TAP Controller Reset (negative assertion)	—		—
3	CKI	I	Input Clock	—		—
24	VEC0/IOBIT7	I/O	Vectored Interrupt ID Bit 0/BIO Signal Bit 7	3-state		logic high††
25	VEC1/IOBIT6	I/O	Vectored Interrupt ID Bit 1/BIO Signal Bit 6	3-state		logic high††
26	VEC2/IOBIT5	I/O	Vectored Interrupt ID Bit 2/BIO Signal Bit 5	3-state		logic high††
27	VEC3/IOBIT4	I/O	Vectored Interrupt ID Bit 3/BIO Signal Bit 4	3-state		logic high††
29	IOBIT3	I/O	BIO Signal Bit 3	3-state		configured as input
30	IOBIT2	I/O	BIO Signal Bit 2	3-state		configured as input
31	IOBIT1	I/O	BIO Signal Bit 1	3-state		configured as input
32	IOBIT0	I/O	BIO Signal Bit 0	3-state		configured as input
33	DOEN	I	SSIO Data Output Enable	—		—
41	DI	I	SSIO Data Input	—		—
39	ICK	I/O	SSIO Input Clock	3-state		configured as input
44	OBE	O	SSIO Output Buffer Empty	3-state		logic high

† During and after reset, the internal clock is selected as the CKI input pin and the CKO output pin is selected as the internal clock.
‡ This pin his internal pull-up circuitry.
§ 3-states by JTAG control.
†† The **ioc** register (see Table 54 on page 99) is cleared after reset, including its EBIO field that controls the multiplexing of the VEC0/IOBIT7, VEC1/IOBIT6, VDC2/IOBIT5, and VEC3/IOBIT4 pins. After reset, these pins are configured as the VEC[3:0] outputs and are logic high.
‡‡ If unused, this pin must be pulled low through a 10 kΩ resistor to Vss.
§§ 3-states if RSTB = 0 or **PHIFC**[PCFIG] = 0.

Pin Information (continued)

Table 80. Pin Descriptions (continued)

TQFP Pin	Symbol	Type	Name/Function	Pin State During Reset (RSTB = 0)	Pin State After Reset (RSTB 0 → 1)
42	IBF	O	SSIO Input Buffer Full	3-state	logic low
35	OLD	I/O	SSIO Output Load	3-state	configured as input
40	ILD	I/O	SSIO Input Load	3-state	configured as input
34	DO	O	SSIO Data Output	3-state	3-state
38	OCK	I/O	SSIO Output Clock	3-state	configured as input
79	SYNC‡‡	I	SSIO Bit Counter Sync	—	—
88	EIBF	O	ESIO Input Buffer Full	3-state	logic low
77	EOEB	I	ESIO Data Output Enable	—	—
80	EDO		ESIO Data Output	3-state	3-state
81	EOFS	I	ESIO Output Frame Sync	—	—
83	EOBC	I	ESIO Output Bit Clock	—	—
84	EIBC	I	ESIO Input Bit Clock	—	—
85	EIFS	I	ESIO Input Frame Sync	—	—
87	EDI	I	ESIO Data Input	—	—
89	EOBE	O	ESIO Output Buffer Empty	3-state	logic high
45, 46, 47, 48, 50, 51, 52, 53	PB[15:8]	I/O§§	PHIF16 Parallel I/O Data Bus 15—8	3-state	3-state
57, 58, 59, 60, 62, 63, 64, 65	PB[7:0]	I/O	PHIF16 Parallel I/O Data Bus 7—0	3-state	3-state
68	POBE	O	PHIF16 Output Buffer Empty	3-state	logic high
69	PIBF	O	PHIF16 Input Buffer Full	3-state	logic low
70	PODS	I	PHIF16 Output Data Strobe	—	—
76	PIDS	I	PHIF16 Input Data Strobe	—	—
71	PBSEL	I	PHIF16 Peripheral Byte Select (8-bit external mode)	—	—
74	PSTAT	I	PHIF16 Peripheral Status Register Select	—	—
75	PCSN	I	PHIF16 Peripheral Chip Select Not	—	—
19, 36, 43, 56, 67, 73, 90, 103, 109, 122, 133, 144	VSS	P	Ground	—	—
16, 28, 37, 49, 61, 72, 82, 97, 108, 115, 128, 138	VDD	P	Power Supply	—	—
4	VDDA	P	Analog Power Supply	—	—
1	VSSA	P	Analog Ground	—	—

† During and after reset, the internal clock is selected as the CKI input pin and the CKO output pin is selected as the internal clock.
‡ This pin his internal pull-up circuitry.
§ 3-states by JTAG control.
†† The **ioc** register (see Table 54 on page 99) is cleared after reset, including its EBIO field that controls the multiplexing of the VEC0/IOBIT7, VEC1/IOBIT6, VDC2/IOBIT5, and VEC3/IOBIT4 pins. After reset, these pins are configured as the VEC[3:0] outputs and are logic high.
‡‡ If unused, this pin must be pulled low through a 10 kΩ resistor to VSS.
§§ 3-states if RSTB = 0 or **PHIFC**[PCFIG] = 0.

Pin Information (continued)

Table 80. Pin Descriptions (continued)

TQFP Pin	Symbol	Type	Name/Function	Pin State During Reset (RSTB = 0)	Pin State After Reset (RSTB 0 → 1)
2, 17, 21, 54, 55, 66, 78, 86, 92, 98, 121, 123, 143	NU	—	Not Usable (No external connections are allowed.)	—	—

† During and after reset, the internal clock is selected as the CKI input pin and the CKO output pin is selected as the internal clock.

‡ This pin his internal pull-up circuitry.

§ 3-states by JTAG control.

†† The **ioc** register (see Table 54 on page 99) is cleared after reset, including its EBIO field that controls the multiplexing of the VEC0/IOBIT7, VEC1/IOBIT6, VDC2/IOBIT5, and VEC3/IOBIT4 pins. After reset, these pins are configured as the VEC[3:0] outputs and are logic high.

‡‡ If unused, this pin must be pulled low through a 10 kΩ resistor to Vss.

§§ 3-states if RSTB = 0 or **PHIFC**[PCFIG] = 0.

Signal Descriptions
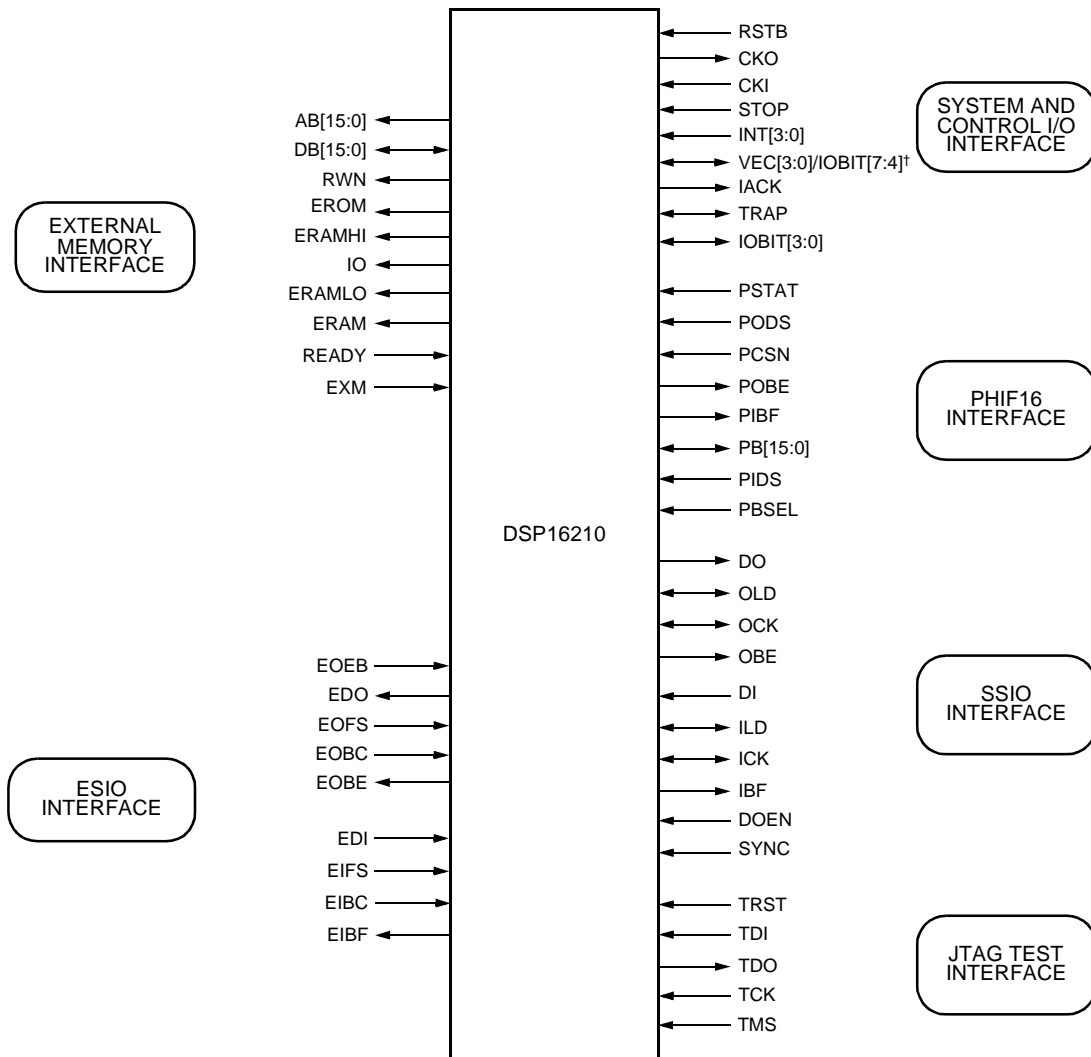
† VEC0 corresponds to IOBIT7, VEC1 corresponds to IOBIT6, VEC2 corresponds to IOBIT5, and VEC3 corresponds to IOBIT4.

Figure 25. DSP16210 Pinout by Interface

Figure 25 shows the interface pinout for the DSP16210. The signals can be separated into six interfaces as shown. Following is a description of these interfaces and the signals that comprise them.

Signal Descriptions (continued)

System Interface and Control I/O Interface

System Interface

The system interface consists of the clock, interrupt, and reset signals for the processor.

RSTB — Device Reset: Negative assertion input. A high-to-low transition causes the processor to enter the reset state. See Reset on page 18 for details.

CKI — Input Clock: The CKI input buffer drives the internal clock (CLK) directly or drives the on-chip PLL (see Clock Synthesis beginning on page 56). The PLL allows the CKI input clock to be at a lower frequency than the internal clock.

STOP — Stop DSP Clocks: Negative assertion input. A high-to-low transition synchronously stops the internal clock, leaving the processor in a defined state. Returning the pin high synchronously restarts the internal clock to continue program execution from where it left off without any loss of state. This hardware feature has the same effect as setting the NOCK bit in the **powerc** register (see Table 65 on page 106).

CKO — Programmable Clock Output: Buffered output clock with options programmable via the **ioc** register (see Table 54 on page 99). The selectable CKO options are as follows:

- CLK: A free-running output clock at the frequency of the internal clock.
- CLKE: Clock at the frequency of the internal clock held high during low-power standby mode (high when AWAIT (**alf**[15]) is high).
- CKI: Clock input pin.
- ZERO: A constant logic 0 output.
- ONE: A constant logic 1 output.

INT[3:0] — External Interrupt Requests: Positive pulse assertion inputs. Hardware interrupt inputs to the DSP16210. Each is enabled via the **inc0** register. When enabled and asserted properly with no equal- or higher-priority interrupts being serviced, each cause the processor to vector to the memory location described in Table 4 on page 21. If an INT pin is asserted for at least the minimum required assertion time (see t22 on page 145), the corresponding external

interrupt request is recorded in the **ins** register. If an INT pin is asserted for less than the minimum required assertion time, the corresponding interrupt request might or might not be recorded in the **ins** register. To avoid erroneous extra entries into the INT interrupt service routine (ISR), an INT pin must be deasserted at least three instruction cycles before the terminating **ireturn** instruction for the associated ISR is executed. When both INT0 and RSTB are asserted, all output and bidirectional pins (except TDO, which 3-states by JTAG control) are put in a 3-state condition.

VEC[3:0] — Vectored Interrupt IDs: Outputs. These four pins indicate which interrupt is currently being serviced by the device. Table 4 on page 21 shows the code associated with each interrupt condition. VEC[3:0] are multiplexed with IOBIT[7:4] (see Pin Multiplexing on page 13). VEC0 corresponds to IOBIT7, VEC1 corresponds to IOBIT6, VEC2 corresponds to IOBIT5, and VEC3 corresponds to IOBIT4. VEC[3:0] defaults to 0xF (all ones) if no interrupt or trap is currently being serviced.

IACK — Interrupt Acknowledge: Positive assertion output. IACK signals when an interrupt is being serviced by the DSP16210. IACK is asserted for three DSP clock cycles.

TRAP — TRAP/Breakpoint Indication: Positive pulse assertion input/output. When asserted, the processor is put into the trap condition, which normally causes a branch to the location **vbase** + 4. Although normally an input, the pin can be configured as an output by the HDS block. As an output, the pin can be used to signal an HDS breakpoint in a multiple processor environment.

Control I/O Interface

The control I/O interface is used for status and control operations provided by the BIO unit.

IOBIT[7:0] — BIO Signals: Input/Output. Each of these pins can be independently configured as either an input or an output. As outputs, they can be independently set, toggled, or cleared. As inputs, they can be tested independently or in combinations for various data patterns. IOBIT[7:4] are pin multiplexed with the VEC[3:0] pins (see Pin Multiplexing on page 13). Setting the EBIO bit in the **ioc** register (bit 8) provides a full 8-bit BIO interface at the associated pins.

Signal Descriptions (continued)

External Memory Interface

The EMI is used to interface the DSP16210 to external memory and I/O devices. It supports read/write operations from/to X- and Y-memory spaces. The interface supports four external memory segments (EROM, ERAMHI, ERAMLO, and IO). The access times for these segments are programmable in the **mwait** register (see Table 58 on page 101).

AB[15:0] — External Memory Address Bus: Output. This 16-bit bus supplies the address for read or write operations to the external memory or I/O. If external memory is not being accessed, AB[15:0] retains its value from the last valid external access.

DB[15:0] — External Memory Data Bus: Input/Output. This 16-bit bidirectional data bus is used for read or write operations to the external memory or I/O. Write data activation can be delayed by setting WDDLY (bit 10 of the **ioc** register).

EXM — External Memory Boot Select: Input. This signal is latched into the device on the rising edge of RSTB. The value of EXM latched in determines which memory region (EROM/IROM) is used when the DSP16210 boots up in response to a device reset. If EXM is low, the DSP16210 boots from IROM. If EXM is high, the DSP16210 boots from EROM.

RWN — EMI Read/Write Not Indicator: Output. When a logic 1, this pin indicates that the data memory (Y) access is a read operation. When a logic 0, it indicates that the memory access is a write operation. This signal can be advanced by setting RWNADV (bit 3 of **ioc**).

EROM — External ROM Enable: Negative assertion output. When asserted, this signal indicates an access (either X or Y) to the EROM segment (see Figure 5 on page 25 or Figure 6 on page 26). The leading edge is delayed by one CLK phase by setting the DENB0 field (bit 0 of **ioc**).

ERAMHI — External RAM High Enable: Negative assertion output. When asserted, this signal indicates a Y access to the external ERAMHI segment (see Figure 6 on page 26). The leading edge is delayed by one CLK phase by setting the DENB1 field (bit 1 of **ioc**).

ERAMLO — External RAM Low Enable: Negative assertion output. When asserted, this signal indicates a Y access to the external ERAMLO segment (see Figure 6 on page 26). The leading edge is delayed by one CLK phase by setting the DENB1 field (bit 1 of **ioc**).

IO — External I/O Enable: Negative assertion output. When asserted, this signal indicates an access to the external data memory-mapped IO segment (see Figure 6 on page 26). The leading edge is delayed by one CLK phase by setting the DENB2 field (bit 2 of **ioc**).

ERAM — External RAM Enable: Negative assertion output. When asserted, this signal indicates an access to either the ERAMHI or ERAMLO external memory segments. The leading edge is delayed by one CLK phase by setting the DENB1 field (bit 1 of **ioc**).

READY — External Memory Access Acknowledge: Negative assertion input. The READY input pin permits an external device to extend the length of an EMI access cycle. The READY pin can be used if an external memory requires an access time greater than 15 cycles, the maximum value programmable in **mwait**. The DSP16210 internally synchronizes the READY pin to the processor clock (CLK). READY must be asserted at least five cycles (plus a setup time) prior to the end of the external memory operation. The DSP16210 adds the number of cycles that READY is asserted to the access time. The appropriate I/O access time field in **mwait** must be four or greater, or the READY pin is ignored.

ESIO Interface

The enhanced serial input/output port (ESIO) is a programmable, hardware-managed, double-buffered input/output port designed to support glueless multichannel I/O processing on a TDM (time-division multiplex) highway. The ESIO communicates the input and output buffer status to the core using input buffer full (EIBF), output buffer empty (EOBE), input frame error (EIFE), output frame error (EOFE), and output collision (ECOL) interrupts. The ESIO external pin signals are described below:

EDI — ESIO Data Input: Serial data is latched on the falling edge of EIBC. Input is at CMOS level and has typically 0.7 V hysteresis.

EIFS — ESIO Input Frame Sync: Input. EIFS defines the beginning of a new input frame (frame mode) or serial data packet (simple mode). To suit a variety of system design requirements, EIFS can be internally inverted and/or delayed for one bit clock (under the control of bits ISLEV and ISDLY of the **ICR** register) to produce the internal input frame sync, IFS. Input is at CMOS level and has typically 0.7 V hysteresis.

Signal Descriptions (continued)

ESIO Interface (continued)

EIBC — ESIO Input Bit Clock: Input. To suit a variety of system design requirements, EIBC can be internally inverted (under the control of the ILEV bit of the **ICR** register) to produce the internal input bit clock, IBC. Input is at CMOS level and has typically 0.7 V hysteresis.

EIBF — ESIO Input Buffer Full: Positive assertion output. When EIBF is high, the serial input buffer is full.

EDO — ESIO Data Output: Serial data (LSB-first) is driven onto EDO on the rising edge of OBC. The rising edge OFS indicates that the first bit of the serial output stream is driven onto the EDO pin on the next rising edge of OBE. EDO is programmed via **OCR**[5] to be either an open-drain output driver or a 3-state output driver. After reset, the driver is in the high-impedance state. This signal is at CMOS level.

EOFS — ESIO Output Frame Sync: Input. EOFS defines the beginning of a new output frame (frame mode) or the beginning of a serial output request (simple mode). To suit a variety of system design requirements, EOFS can be internally inverted (under the control of the OSLEV bit of the **OCR** register) to produce the internal output frame sync, OFS. Input is at CMOS level and has typically 0.7 V hysteresis.

EOBC — ESIO Output Bit Clock: Input. To suit a variety of system design requirements, EOBC can be internally inverted (under the control of **OCR** register bit OLEV) to produce the internal output bit clock, OBC. Input is at CMOS level and has typically 0.7 V hysteresis.

EOEB — ESIO Data Output Enable: Negative assertion input. When EOEB is high, EDO is forced into high impedance. Input is at CMOS level and has typically 0.7 V hysteresis.

EOBE — ESIO Output Buffer Empty: Positive assertion output. When EOBE is high, the serial output buffer is empty.

SSIO Interface

The SSIO interface pins implement a full-featured serial I/O channel.

DI — SSIO Data Input: Serial data is latched on the rising edge of ICK, either LSB or MSB first, according to the **SSIOC** register MSB field (see Table 70 on page 110).

ICK — SSIO Input Clock: Input/Output. The clock for serial input data. In active mode, ICK is an output; in passive mode, ICK is an input, according to the **SSIOC** register ICK field (see Table 70 on page 110). Input has typically 0.7 V hysteresis.

ILD — SSIO Input Load: Input/Output. The clock for loading the input buffer. A falling edge of ILD indicates the beginning of a serial input word. In active mode, ILD is an output; in passive mode, ILD is an input, according to the **SSIOC** register ILD field (see Table 70 on page 110). Input has typically 0.7 V hysteresis.

IBF — SSIO Input Buffer Full: Positive assertion output. IBF is asserted when the input register, **SSDX**(in), is filled. IBF is cleared when MIOU1 transfers **SSDX**(in) to IORAM1. IBF is also cleared by asserting RSTB.

DO — SSIO Data Output: The serial data output either LSB or MSB first (according to the **SSIOC** register MSB field). DO normally changes on the rising edges of OCK but can be programmed to change on falling edges, as determined by the DODLY field of the **SSIOC** register. DO is 3-stated when DOEN is high.

DOEN — SSIO Data Output Enable: Negative assertion input. DO is enabled only if DOEN is low.

OCK — SSIO Output Clock: Input/Output. The clock for serial output data. In active mode, OCK is an output; in passive mode, OCK is an input, according to the **SSIOC** register OCK field (see Table 70 on page 110). Input has typically 0.7 V hysteresis.

OLD — SSIO Output Load: Input/Output. A falling edge of OLD indicates the beginning of a serial output word. In active mode, OLD is an output; in passive, OLD is an input, according to the **SSIOC** register OLD field (see Table 70 on page 110). Input has typically 0.7 V hysteresis.

OBE — SSIO Output Buffer Empty: Positive assertion output. OBE is asserted when the output register, **SSDX**(out), is emptied (moved to the output shift register for transmission). OBE is cleared when MIOU1 fills **SSDX**(out).

SYNC — SSIO Bit Counter Sync: Input. A falling edge of SYNC causes the resynchronization of the active ILD and OLD generators. Input has typically 0.7 V hysteresis. If unused, this pin must be pulled low through a 10 kΩ resistor to Vss.

Signal Descriptions (continued)

PHIF16 Interface

The PHIF16 interface implements a full 16-bit host interface to standard microprocessors.

PB[15:0] — PHIF16 Parallel I/O Data Bus: Input/Output. This 16-bit bidirectional bus is used to input data to, or output data from, the PHIF16. It can be configured as an 8-bit external bus where PB[15:8] are 3-stated.

PCSN — PHIF16 Peripheral Chip Select Not: Negative assertion input. If PCSN is low, the data strobes PIDS and PODS are enabled. If PCSN is high, the DSP16210 ignores any activity on PIDS and PODS.

PBSEL — PHIF16 Peripheral Byte Select: Input. The assertion level is configurable in software via **PHIFC**[3]. Selects the high or low byte of PDX available for host accesses (8-bit external mode).

PSTAT — PHIF16 Peripheral Status Register Select: Input. If a logic 0, the PHIF16 outputs the PDX(out) register on the PB bus. If a logic 1, the PHIF16 outputs the contents of the **PSTAT** register on PB[7:0].

PIDS — PHIF16 Input Data Strobe: Input. Supports either *Intel* or *Motorola* protocols. Configured by the **PHIFC**[PSTROBE] control register bit.

In *Intel* mode: Negative assertion. PIDS is pulled low by an external device to indicate that data is available on the PB bus. The DSP latches data on the PB bus on the rising edge (low-to-high transition) of PIDS or PCSN, whichever comes first.

In *Motorola* mode: PIDS (PRWN) functions as a read/write strobe. The external device sets PIDS (PRWN) to a logic 0 to indicate that data is available on the PB bus (write operation by the external device). A logic 1 on PIDS (PRWN) indicates an external read operation by the external device.

PODS — PHIF16 Output Data Strobe: Input. Software-configurable to support both *Intel* and *Motorola* protocols:

In *Intel* mode: Negative assertion. When PODS is pulled low by an external device, the DSP16210 places the contents of the parallel output register, PDX(out), onto the PB bus.

In *Motorola* mode: Software-configurable assertion level. The external device uses PODS (PDS) as its data strobe for both read and write operations.

PIBF — PHIF16 Input Buffer Full: Output. The assertion level is configurable in software (**PHIFC**[4]). This flag is cleared after reset, indicating an empty input register PDX(in). PIBF is set immediately after the rising edge of PIDS or PCSN, indicating that data has been latched into the PDX(in) register. When the DSP16210 reads the contents of this register, emptying the buffer, this flag is cleared. Configured in software (**PHIFC**[5]), PIBF can become the logical OR of the PIBF and POBE flags.

POBE — PHIF16 Output Buffer Empty: Output. The assertion level is configurable in software (**PHIFC**[4]). This flag is set after reset, indicating an empty output register PDX(out). POBE is set immediately after the rising edge of PODS or PCSN, indicating that the data in PDX(out) has been driven onto the PB bus. When the DSP16210 writes to PDX(out), filling the buffer, this flag is cleared.

JTAG Test Interface

The JTAG test interface has features that allow programs and data to be downloaded into the DSP via four pins. This provides extensive test and diagnostic capability. In addition, internal circuitry allows the device to be controlled through the JTAG port to provide on-chip, in-circuit emulation. Lucent Technologies provides hardware and software tools to interface to the on-chip HDS via the JTAG port.

Note: The DSP16210 provides all JTAG/*IEEE* 1149.1 standard test capabilities including boundary scan.

TDI — JTAG Test Data Input: Serial input signal. All serial-scanned data and instructions are input on this pin. This pin has an internal pull-up resistor.

TDO — JTAG Test Data Output: Serial output signal. Serial-scanned data and status bits are output on this pin.

TMS — JTAG Test Mode Select: Mode control signal that, when combined with TCK, controls the scan operations. This pin has an internal pull-up resistor.

TCK — JTAG Test Clock: Serial shift clock. This signal clocks all data into the port through TDI, and out of the port through TDO, and controls the port by latching the TMS signal inside the state-machine controller.

TRST — JTAG TAP Controller Reset: Negative assertion. Test reset. When asserted low, resets JTAG TAP controller. In an application environment, this pin must be asserted prior to or concurrent with RSTB.

DSP16210 Boot Routines

There are many subroutines in the IROM of the DSP16210 that allow the user to perform various functions. The primary function of these routines is to download code and data to the internal DPRAM and external memory from the PHIF16 or EMI ports. Other functions include memory test routines, a routine to enable or disable the PLL, and reserved production test routines. Once the downloads and/or other functions are completed, the user can select a boot routine that branches to the beginning of either the DPRAM or EROM memory segment.

After reset with the EXM pin low, the DSP16210 begins executing from location 0x20000 in IROM, which contains a branch to location 0x20400. The program at this location configures the parallel host interface port, PHIF16, for a single, 8-bit *Intel* style transfer (refer to Parallel Host Interface (PHIF16) beginning on page 49), enables INT3, and leaves the PLL disabled. The program then waits for a host device interfaced to the PHIF16 port to write an 8-bit command onto PB[7:0] and strobe PIDS. The program inputs this value and decodes it to select the appropriate boot routine. To terminate any routine, the host can assert the INT3 pin; otherwise, the routine completes its function.

At the completion or termination (interruption) of a routine, a value of 0xED is written out the PHIF16 port. In response to the POBE flag, the host must read this value from the port in order for the boot code to continue. The program flow is stalled until this port is read. After this handshake byte has been read, the PHIF16 port is returned to 8-bit *Intel* style mode and waits for the next command.

Once the user has completed the routines of interest, e.g., download of code and data or tested memory, the host can direct the DSP16210 to begin execution of user code from either the DPRAM (location 0x00000) or EROM (location 0x80000) XMAP locations.

Note: Upon exiting the boot code, the following core registers are not reinitialized to their reset states as defined in the *DSP16000 Digital Signal Processor Core* Information Manual: **inc0**, **rb0**, **re0**, **vbase**, **cloop**, **cstate**. With the exception of the **ioc** and **PHIFC** registers, none of the peripheral registers are reinitialized to their reset states as defined in Table 78 on page 114. It is recommended that the user code immediately globally disable interrupts by executing a **di** instruction and clear all pending interrupts by clearing **ins** (ins = 0xfffff).

DSP16210 Boot Routines (continued)

Commands

Table 81 is a summary of the 8-bit PHIF16 command codes and their associated boot routines. Boot routines that use external memory first configure the **mwait** register as shown in the table. Boot routines that use the PHIF16 port configure the port as shown in the table.

Table 81. Command Encoding for Boot Routines

Command Code	mwait Setting	Function/Download						
		Download From	Download To	Configuration				
				PHIF16[†]				MIOU0 DMA Block[‡]
				External Bus	Logical Transfer	Mode	PODS/ PDS	Byte- Swapping	
0x18	0x0030	ERAMHI[§]	DPRAM (60K)	—	—	—	—	—	—
0x58		ERAMLO[§]							
0x98		ERAMLO[§]							
0x19	0x0111	IO[§]							
0x59	0x0222								
0x99	0x0444								
0xD9	0x0FFF								
0x1B	0x0111	EROM[§]							
0x5B	0x0222								
0x9B	0x0444								
0xDB	0x0FFF								
0x01	—	PHIF16	DPRAM (60K)	8-bit	8-bit	*Motorola*	active-low	—	1 word
0x02							active-high		
0x03						*Intel*	active-low		
0x06				8-bit	16-bit	*Motorola*	active-low	No	
0x07							active-high		
0x08						*Intel*	active-low		
0x09				16-bit	16-bit	*Motorola*	active-low	No	
0x0A							active-high		
0x0B						*Intel*	active-low		
0x0C						*Motorola*	active-low	No	64 words
0x0D							active-high		
0x0E						*Intel*	active-low		
0x16						*Motorola*	active-low	Yes	1 word
0x17							active-high		
0x1C						*Motorola*	active-low	Yes	64 words
0x1D							active-high		
0x0F						*Motorola*	active-low	No	512 words
0x10							active-high		
0x11						*Intel*	active-low		

† The boot routine configures the PHIF16 by writing to the **PHIFC** register. Specifically, the external bus configuration, logical transfer size, and byte swapping are controlled by the PMODE and PCFIG fields, the mode is controlled by the PSTROBE field, and the PODS/PDS active-low/high configuration is controlled by the PSTRB field. After the download is complete, the boot routine returns the PHIF16 to its initial state (configures it for *Intel* mode with an 8-bit external bus and 8-bit logical transfers).

‡ This is the size of each input DMA transfer that the boot routine directs the MIOU0 to perform without core intervention. MIOU0 performs input DMA from the PHIF16 block. The boot routine configures the input block size by programming the **ILEN0** register.

§ The first 60 Kword locations of the segment are copied into the DPRAM.

DSP16210 Boot Routines (continued)

Commands (continued)

Table 81. Command Encoding for Boot Routines (continued)

Command Code	mwait Setting	Download From	Download To	External Bus	Logical Transfer	Mode	PODS/PDS	Byte-Swapping	MIOU0 DMA Block‡
				PHIF16†					
0x20	0x0111	PHIF16	EROM (64K)	8-bit	8-bit	Motorola	active-low	—	1 word
0x21							active-high		
0x22						Intel	active-low		
0x23				8-bit	16-bit	Motorola	active-low	No	
0x24							active-high		
0x25						Intel	active-low		
0x26				16-bit	16-bit	Motorola	active-low	No	
0x27							active-high		
0x28						Intel	active-low		
0x60	0x0222			8-bit	8-bit	Motorola	active-low	—	
0x61							active-high		
0x62						Intel	active-low		
0x63				8-bit	16-bit	Motorola	active-low	No	
0x64							active-high		
0x65						Intel	active-low		
0x66				16-bit	16-bit	Motorola	active-low	No	
0x67							active-high		
0x68						Intel	active-low		
0xA0	0x0444			8-bit	8-bit	Motorola	active-low	—	
0xA1							active-high		
0xA2						Intel	active-low		
0xA3				8-bit	16-bit	Motorola	active-low	No	
0xA4							active-high		
0xA5						Intel	active-low		
0xA6				16-bit	16-bit	Motorola	active-low	No	
0xA7							active-high		
0xA8						Intel	active-low		
0xE0	0x0FFF			8-bit	8-bit	Motorola	active-low	—	
0xE1							active-high		
0xE2						Intel	active-low		
0xE3				8-bit	16-bit	Motorola	active-low	No	
0xE4							active-high		
0xE5						Intel	active-low		
0xE6				16-bit	16-bit	Motorola	active-low	No	
0xE7							active-high		
0xE8						Intel	active-low		

† The boot routine configures the PHIF16 by writing to the **PHIFC** register. Specifically, the external bus configuration, logical transfer size, and byte swapping are controlled by the PMODE and PCFIG fields, the mode is controlled by the PSTROBE field, and the PODS/PDS active-low/high configuration is controlled by the PSTRB field. After the download is complete, the boot routine returns the PHIF16 to its initial state (configures it for *Intel* mode with an 8-bit external bus and 8-bit logical transfers).

‡ This is the size of each input DMA transfer that the boot routine directs the MIOU0 to perform without core intervention. MIOU0 performs input DMA from the PHIF16 block. The boot routine configures the input block size by programming the **ILEN0** register.

§ The first 60 Kword locations of the segment are copied into the DPRAM.

DSP16210 Boot Routines (continued)

Commands (continued)

Table 81. Command Encoding for Boot Routines (continued)

Command Code	mwait Setting	Download From	Download To	External Bus	Logical Transfer	Mode	PODS/ PDS	Byte-Swapping	MIOU0 DMA Block‡
				Function/Download					
		Download From	Download To	Configuration					MIOU0 DMA Block‡
				PHIF16†					
				External Bus	Logical Transfer	Mode	PODS/ PDS	Byte-Swapping	
0x29	0x0111	PHIF16	EROM (64K)	16-bit	16-bit	*Motorola*	active-low	No	64 words
0x2A							active-high		
0x2B						*Intel*	active-low		
0x2E						*Motorola*	active-low	Yes	1 word
0x2F							active-high		
0x32						*Motorola*	active-low	Yes	64 words
0x34							active-high		
0x35						*Motorola*	active-low	No	512 words
0x36							active-high		
0x37						*Intel*	active-low		
0x69	0x0222					*Motorola*	active-low	No	64 words
0x6A							active-high		
0x6B						*Intel*	active-low		
0x6E						*Motorola*	active-low	Yes	1 word
0x6F							active-high		
0x72						*Motorola*	active-low	Yes	64 words
0x74							active-high		
0x75						*Motorola*	active-low	No	512 words
0x76							active-high		
0x77						*Intel*	active-low		
0xA9	0x0444					*Motorola*	active-low	No	64 words
0xAA							active-high		
0xAB						*Intel*	active-low		
0xAE						*Motorola*	active-low	Yes	1 word
0xAF							active-high		
0xB2						*Motorola*	active-low	Yes	64 words
0xB4							active-high		
0xB5						*Motorola*	active-low	No	512 words
0xB6							active-high		
0xB7						*Intel*	active-low		

† The boot routine configures the PHIF16 by writing to the **PHIFC** register. Specifically, the external bus configuration, logical transfer size, and byte swapping are controlled by the PMODE and PCFIG fields, the mode is controlled by the PSTROBE field, and the PODS/PDS active-low/high configuration is controlled by the PSTRB field. After the download is complete, the boot routine returns the PHIF16 to its initial state (configures it for *Intel* mode with an 8-bit external bus and 8-bit logical transfers).

‡ This is the size of each input DMA transfer that the boot routine directs the MIOU0 to perform without core intervention. MIOU0 performs input DMA from the PHIF16 block. The boot routine configures the input block size by programming the **ILEN0** register.

§ The first 60 Kword locations of the segment are copied into the DPRAM.

DSP16210 Boot Routines (continued)

Commands (continued)

Table 81. Command Encoding for Boot Routines (continued)

Command Code	mwait Setting	Download From	Download To	External Bus	Logical Transfer	Mode	PODS/ PDS	Byte- Swapping	MIOU0 DMA Block‡
				Function/Download					
				Configuration					
				PHIF16†					
0xE9	0x0FFF	PHIF16	EROM (64K)	16-bit	16-bit	Motorola	active-low	No	64 words
0xEA							active-high		
0xEB						Intel	active-low		
0xEE						Motorola	active-low	Yes	1 word
0xEF							active-high		
0xF2						Motorola	active-low	Yes	64 words
0xF4							active-high		
0xF5						Motorola	active-low	No	512 words
0xF6							active-high		
0xF7						Intel	active-low		
0x12	0x0030	PHIF16	ERAMLO & ERAMHI (128K)	16-bit	16-bit	Motorola	active-high	No	512 words
0x52							active-low		
0x92						Intel	active-low		
0x13	0x0030		ERAMLO			Motorola	active-high		
0x53							active-low		
0x93						Intel	active-low		
0x14	0x0030		ERAMHI			Motorola	active-high		
0x54							active-low		
0x94						Intel	active-low		
0x39	0x0111		IO			Motorola	active-low		
0x3B							active-high		
0x3D						Intel	active-low		
0x79	0x0222					Motorola	active-low		
0x7B							active-high		
0x7D						Intel	active-low		
0xB9	0x0444					Motorola	active-low		
0xBB							active-high		
0xBD						Intel	active-low		
0xF9	0x0FFF					Motorola	active-low		
0xFB							active-high		
0xFD						Intel	active-low		
0x04	—	Test internal 1K IORAM0 memory segment—write result word to **ar3** (0x0FAB for passed and 0x0BAD for failed).							
0x05	—	Test internal 1K IORAM1 memory segment—write result word to **ar3** (0x0FAB for passed and 0x0BAD for failed).							
0x30	—	Test internal 60K DPRAM memory segment—write result word to **ar3** (0x0FAB for passed and 0x0BAD for failed).							

† The boot routine configures the PHIF16 by writing to the **PHIFC** register. Specifically, the external bus configuration, logical transfer size, and byte swapping are controlled by the PMODE and PCFIG fields, the mode is controlled by the PSTROBE field, and the PODS/PDS active-low/high configuration is controlled by the PSTRB field. After the download is complete, the boot routine returns the PHIF16 to its initial state (configures it for *Intel* mode with an 8-bit external bus and 8-bit logical transfers).

‡ This is the size of each input DMA transfer that the boot routine directs the MIOU0 to perform without core intervention. MIOU0 performs input DMA from the PHIF16 block. The boot routine configures the input block size by programming the **ILEN0** register.

§ The first 60 Kword locations of the segment are copied into the DPRAM.

DSP16210 Boot Routines (continued)

Commands (continued)

Table 81. Command Encoding for Boot Routines (continued)

Command Code	mwait Setting	Function/Download							
		Download From	Download To	Configuration					MIOU0 DMA Block‡
				PHIF16†					
				External Bus	Logical Transfer	Mode	PODS/ PDS	Byte-Swapping	
0x70	—	Test internal CACHE (62 words) memory segment—write result word to **ar3** (0x0FAB for passed and 0x0BAD for failed).							
0x31	0x0111	Test external 64K ERAMLO memory segment—write result word to **ar3** (0x0FAB for passed and 0x0BAD for failed).							
0x71	0x0222								
0xB1	0x0444								
0xF1	0x0FFF								
0x33	0x0111	Test external 64K ERAMHI memory segment—write result word to **ar3** (0x0FAB for passed and 0x0BAD for failed).							
0x73	0x0222								
0xB3	0x0444								
0xF3	0x0FFF								
0x1A	0x0111	Test external 128K ERAMLO and ERAMHI memory segments—write result word to **ar3** (0x0FAB for passed and 0x0BAD for failed).							
0x5A	0x0222								
0x9A	0x0444								
0xDA	0x0FFF								
0x38	0x0111	Test external 64K EROM memory segment—write result word to **ar3** (0x0FAB for passed and 0x0BAD for failed).							
0x78	0x0222								
0xB8	0x0444								
0xF8	0x0FFF								
0x3F	0x0111	Test external 64K IO memory segment—write result word to **ar3** (0x0FAB for passed and 0x0BAD for failed).							
0x7F	0x0222								
0xBF	0x0444								
0xFF	0x0FFF								
0x2C	—	Perform checksum on 60K DPRAM. Write the checksum result (two 16-bit words) to the PHIF16 port in *Intel* 16-bit mode. The purpose of the checksum routine is to check that code has been downloaded properly into the DPRAM segment. After the host reads the two checksum words, return the PHIF16 to *Intel* 8-bit mode and write the handshake byte 0xED to the PHIF16 port.							
0x2D	0x0111	Perform checksum on 64K EROM. Write the checksum result (two 16-bit words) to the PHIF16 port in *Intel* 16-bit mode. The purpose of the checksum routine is to check that code has been downloaded properly into the EROM segment. After the host reads the two checksum words, return the PHIF16 to *Intel* 8-bit mode and write the handshake byte 0xED to the PHIF16 port.							
0x6D	0x0222								
0xAD	0x0444								
0xED	0x0FFF								
0x15	—	Copy word in **ar3** (result of the previously run memory test—0x0FAB for passed and 0x0BAD for failed) to the PHIF16 port in *Motorola* 16-bit mode with active-high PDS.							
0x55	—	Copy word in **ar3** (result of the previously run memory test—0x0FAB for passed and 0x0BAD for failed) to the PHIF16 port in *Motorola* 16-bit mode with active-low PDS.							
0x95	—	Copy word in **ar3** (result of the previously run memory test—0x0FAB for passed and 0x0BAD for failed) to the PHIF16 port in *Intel* 16-bit mode.							
0xD5	—	Copy word in **ar3** (result of the previously run memory test—0x0FAB for passed and 0x0BAD for failed) to the PHIF16 port in *Motorola* 16-bit mode with active-high PDS.							

† The boot routine configures the PHIF16 by writing to the **PHIFC** register. Specifically, the external bus configuration, logical transfer size, and byte swapping are controlled by the PMODE and PCFIG fields, the mode is controlled by the PSTROBE field, and the PODS/PDS active-low/high configuration is controlled by the PSTRB field. After the download is complete, the boot routine returns the PHIF16 to its initial state (configures it for *Intel* mode with an 8-bit external bus and 8-bit logical transfers).

‡ This is the size of each input DMA transfer that the boot routine directs the MIOU0 to perform without core intervention. MIOU0 performs input DMA from the PHIF16 block. The boot routine configures the input block size by programming the **ILEN0** register.

§ The first 60 Kword locations of the segment are copied into the DPRAM.

DSP16210 Boot Routines (continued)

Commands (continued)

Table 81. Command Encoding for Boot Routines (continued)

Command Code	mwait Setting	Function/Download							
		Download From	Download To	Configuration					
				PHIF16†					MIOU0 DMA Block‡
				External Bus	Logical Transfer	Mode	PODS/ PDS	Byte-Swapping	
0x3A	—	Reserved for production test.							
0x7A	—								
0xBA	—								
0x1F	—								
0x5F	—								
0x9F	—								
0xDF	—								
0x1E	—	Disable PLL; CLK = CKI.							
0x5E	—	Enable PLL; CLK = 2 × CKI (25 MHz ≤ CKI ≤ 50 MHz).							
0x9E	—	Enable PLL; CLK = 5 × CKI (10 MHz ≤ CKI ≤ 20 MHz).							
0xDE	—	Enable PLL; CLK = 10 × CKI (5 MHz ≤ CKI ≤ 10 MHz).							
0x3C	0x0111	Execute from EROM (branch to location 0x80000).							
0x7C	0x0222								
0xBC	0x0444								
0xFC	0x0FFF								
0x3E	—	Execute from DPRAM (branch to location 0x00000).							

† The boot routine configures the PHIF16 by writing to the **PHIFC** register. Specifically, the external bus configuration, logical transfer size, and byte swapping are controlled by the PMODE and PCFIG fields, the mode is controlled by the PSTROBE field, and the PODS/PDS active-low/high configuration is controlled by the PSTRB field. After the download is complete, the boot routine returns the PHIF16 to its initial state (configures it for *Intel* mode with an 8-bit external bus and 8-bit logical transfers).

‡ This is the size of each input DMA transfer that the boot routine directs the MIOU0 to perform without core intervention. MIOU0 performs input DMA from the PHIF16 block. The boot routine configures the input block size by programming the **ILEN0** register.

§ The first 60 Kword locations of the segment are copied into the DPRAM.

Device Characteristics

Absolute 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.

External leads can be bonded and soldered safely at temperatures of up to 300 °C.

Table 82. Absolute Maximum Ratings

Parameter	Min	Max	Unit
Voltage Range on V_{DD} or V_{DDA} with Respect to Ground	–0.5	+4.6	V
Voltage Range on Any Signal Pin	V_{SS} – 0.5	V_{DD} + 0.5	V
Power Dissipation	—	1	W
Junction Temperature (T_J)	–40	+125	°C
Storage Temperature Range	–65	+150	°C

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. Lucent Technologies employs a human-body model for ESD-susceptibility testing. Since the failure voltage of electronic devices is dependent on the current, 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 Lucent Technologies human-body model test circuit. The breakdown voltage for the DSP16210 is greater than 2000 V.

Recommended Operating Conditions

Table 83. Recommended Operating Conditions

Maximum Internal Clock (CLK) Frequency	Minimum Internal Clock (CLK) Period T	Package	Supply Voltage V_{DD} (V)		Ambient Temperature T_A (°C)	
			Min	Max	Min	Max
100 MHz	10 ns	TQFP	3.0	3.6	–40	85

The ratio of the instruction cycle rate to the input clock frequency is 1:1 without the PLL and M/2N with the PLL selected (see Clock Synthesis beginning on page 56). **The maximum input clock (CKI pin) frequency is 50 MHz. The PLL must be used when an internal clock frequency greater than 50 MHz is required.**

Device Characteristics (continued)

Recommended Operating Conditions (continued)

Package Thermal Considerations

The recommended operating temperature previously specified is based on the maximum power, package type, and maximum junction temperature. The following equations describe the relationship between these parameters. If the applications' maximum power is less than the worst-case value, this relationship determines a higher maximum ambient temperature or the maximum temperature measured at top dead center of the package.

$T_A = T_J - P \times \Theta_{JA}$

$T_{TDC} = T_J - P \times \Theta_{J\text{-}TDC}$

where T_A is the still-air-ambient temperature and T_{TDC} is the temperature measured by a thermocouple at the top dead center of the package.

Table 84. Package Thermal Considerations

Device Package	Parameter	Value	Unit
144-pin TQFP	Maximum Junction Temperature (T_J)	115	°C
144-pin TQFP	Maximum Thermal Resistance in Still-Air-Ambient (Θ_{JA})	29.5	°C/W
144-pin TQFP	Maximum Thermal Resistance, Junction to Top Dead Center ($\Theta_{J\text{-}TDC}$)	4	°C/W

WARNING: Due to package thermal constraints, proper precautions in the user's application should be taken to avoid exceeding the maximum junction temperature of 115 °C. Otherwise, the device performance is adversely affected.

Electrical Characteristics and Requirements

Electrical characteristics refer to the behavior of the device under specified conditions. Electrical requirements refer to conditions imposed on the user for proper operation of the device. The parameters below are valid for the conditions described in the previous section, Recommended Operating Conditions.

Table 85. Electrical Characteristics and Requirements

Parameter	Symbol	Min	Max	Unit

Input Voltage:
    Low
    High
	V_{IL}
	V_{IH}
		−0.3
		0.7 * V_{DD}
			0.3 * V_{DD}
			V_{DD} + 0.3
				V
				V

Input Current (except TMS, TDI):
    Low (V_{IL} = 0 V, V_{DD} = 3.6 V)
    High (V_{IH} = 3.6 V, V_{DD} = 3.6 V)
	I_{IL}
	I_{IH}
		−5
		—
			—
			5
				µA
				µA

Input Current (TMS, TDI):
    Low (V_{IL} = 0 V, V_{DD} = 3.6 V)
    High (V_{IH} = 3.6 V, V_{DD} = 3.6 V)
	I_{IL}
	I_{IH}
		−100
		—
			—
			5
				µA
				µA

Output Low Voltage:
    Low (I_{OL} = 2.0 mA)
    Low (I_{OL} = 50 µA)
	V_{OL}
	V_{OL}
		—
		—
			0.4
			0.2
				V
				V

Output High Voltage:
    High (I_{OH} = −2.0 mA)
    High (I_{OH} = −50 µA)
	V_{OH}
	V_{OH}
		V_{DD} − 0.7
		V_{DD} − 0.2
			—
			—
				V
				V

Output 3-State Current:
    Low (V_{DD} = 3.6 V, V_{IL} = 0 V)
    High (V_{DD} = 3.6 V, V_{IH} = 3.6 V)
	I_{OZL}
	I_{OZH}
		−10
		—
			—
			10
				µA
				µA

Input Capacitance	CI	—	5	pF

Electrical Characteristics and Requirements (continued)
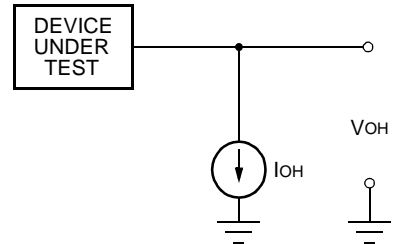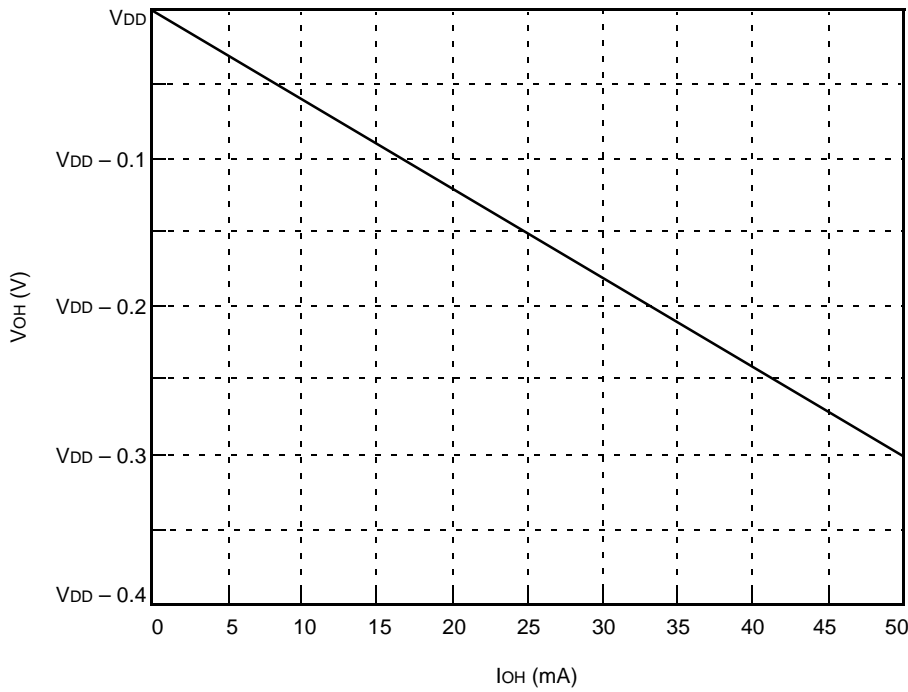
5-4007(C).a

Figure 26. Plot of V_{OH} vs. I_{OH} Under Typical Operating Conditions
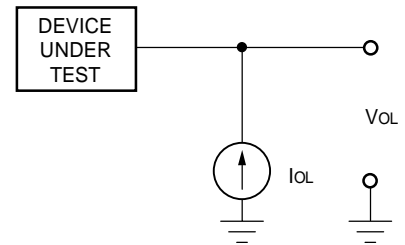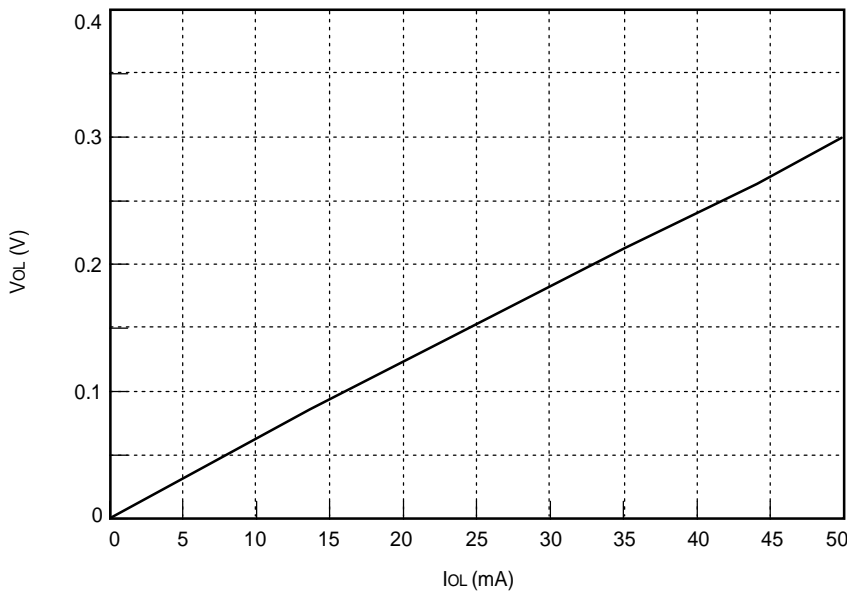
5-4008(C).b

Figure 27. Plot of V_{OL} vs. I_{OL} Under Typical Operating Conditions

Electrical Characteristics and Requirements (continued)

Power Dissipation

Power dissipation is highly dependent on DSP program activity and the frequency of operation. The typical power dissipation listed is for a selected application.

Table 86. Power Dissipation

Condition[†]		Typical Power Dissipation (mW)	
		Peripherals On (powerc = 0x0000)	Peripherals Off (powerc = 0x181F)
Normal Operation	PLL Disabled[‡] and Deselected[§] CLK = CKI = 40 MHz	327	315
	PLL Enabled[‡] and Selected[§] **pllc** = 0xEC0E CKI = 10 MHz, CLK = 40 MHz	340	328
Low-Power Standby Mode (AWAIT (**alf**[15]) = 1)	PLL Disabled[‡] and Deselected[§] CLK = CKI = 40 MHz	51	39
	PLL Enabled[‡] and Selected[§] **pllc** = 0xEC0E CKI = 10 MHz, CLK = 40 MHz	64	52

† In all cases, V_{DD} = V_{DDA} = 3.0 V, unused inputs are tied to V_{DD} or V_{SS}, and the CKO output pin is held low (**ioc** = 0x0040).
‡ The PLL is disabled (powered down) if the PLLEN field (**pllc**[15]) is cleared, which is the default after reset. The PLL is enabled (powered up) if the PLLEN field (**pllc**[15]) is set.
§ The PLL is deselected if the PLLSEL field (**pllc**[14]) is cleared, which is the default after reset. The PLL is selected if the PLLSEL field (**pllc**[14]) is set.

The power dissipation listed is for internal power dissipation only. Total power dissipation can be calculated on the basis of the application by adding C x V_{DD}^2 x f for each output, where C is the additional load capacitance and f is the output frequency. Power dissipation due to the input buffers is highly dependent on the input voltage level. At full CMOS levels, essentially no dc current is drawn. However, for levels between the power supply rails, especially at or near the threshold of $V_{DD}/2$, high currents can flow.

The following recommendations apply:

■ Input and I/O buffers can be left untied with no power dissipation penalty because the input voltage levels of the input and I/O buffers are designed to remain at full CMOS levels when not driven.

■ Unused I/O pins that require a known value (1 or 0) for correct device operation should be tied to V_{DD} or V_{SS} through a 10 kΩ resistor.

■ Unused input pins that require a known value (1 or 0) should be tied to V_{DD} or V_{SS}.

WARNING: The device needs to be clocked for at least six CKI cycles during reset after powerup. Otherwise, high currents might flow.

Timing Characteristics and Requirements

Timing characteristics refer to the behavior of the device under specified conditions. Timing requirements refer to conditions imposed on the user for proper operation of the device. All timing data is valid for the following conditions:

T_A = –40 °C to +85 °C (See Recommended Operating Conditions on page 133.)

V_{DD} = 3.3 V ± 0.3 V, V_{SS} = 0 V (See Recommended Operating Conditions on page 133.)

Capacitance load on outputs (C_L) = 50 pF

Output characteristics can be derated as a function of load capacitance (C_L).

All outputs except CKO: 0.025 ns/pF ≤ dt/dC_L ≤ 0.07 ns/pF for 10 ≤ C_L ≤ 100 pF
CKO: 0.01 ns/pF ≤ dt/dC_L ≤ 0.025 ns/pF for 10 ≤ C_L ≤ 100 pF
at V_{IH} for rising edge and at V_{IL} for falling edge

For example, if the actual load capacitance on a pin other than CKO is 30 pF instead of 50 pF, the maximum derating for a rising edge is (30 – 50) pF x 0.07 ns/pF = 1.4 ns **less** than the specified rise time or delay that includes a rise time. The minimum derating for the same 30 pF load would be (30 – 50) pF x 0.025 ns/pF = 0.5 ns.

Test conditions for inputs:

■ Rise and fall times of 4 ns or less
■ Timing reference levels for delays = V_{IH}, V_{IL}

Test conditions for outputs (unless noted otherwise):

■ C_{LOAD} = 50 pF
■ Timing reference levels for delays = V_{IH}, V_{IL}
■ 3-state delays measured to the high-impedance state of the output driver

Unless otherwise noted, CKO in the timing diagrams is the free-running CKO.

Timing Characteristics and Requirements (continued)

Phase-Lock Loop

Table 87. Frequency Ranges for PLL Output

Parameter	Symbol	Min	Max	Unit
PLL Output[†] Frequency Range (V$_{DD}$ = 3.3 V $\pm$ 0.3 V)	f$_{PLL}$	50	120	MHz
Input Jitter at CKI	—	—	200	ps-rms

† The values of M and N in the **pllc** register (see Table 64 on page 105) must be set so that f$_{PLL}$ is in the appropriate range. Choose the lowest value of N and then the appropriate value of M for
f$_{CLK}$ = f$_{PLL}$ = f$_{CKI}$ $\times$ M/2N.

Table 88. PLL Loop Filter Settings and Lock-In Time

M	pllc[11:8] (LF[3:0])	Typical Lock-In Time (µs)[†]
23—24	1011	30
21—22	1010	30
19—20	1001	30
16—18	1000	30
12—15	0111	30
8—11	0110	30
2—7	0100	30

† Lock-in time is the time following assertion of the PLLEN bit of the **pllc** register during which the PLL output clock is unstable. The DSP must operate from the CKI input clock or from the slow ring oscillator while the PLL is locking. The DSP16210 signals completion of the lock-in interval by setting the LOCK flag.

Timing Characteristics and Requirements (continued)

Wake-Up Latency

Table 89 specifies the wake-up latency for various low-power modes. The wake-up latency is the delay between exiting a low-power mode and resumption of normal execution.

Table 89. Wake-Up Latency

Condition		Wake-Up Latency	
		(PLL Deselected[†] During Normal Execution)	**(PLL Enabled[‡] and Selected[†] During Normal Execution)**
Low-Power Standby Mode (AWAIT (**alf**[15]) = 1)	PLL Disabled[‡] During Standby	$3T^\S$	$3T^\S + t_L{}^{\dagger\dagger}$
	PLL Enabled[‡] During Standby	—	$3T^\S$
Low-Power Standby Mode (AWAIT (**alf**[15]) = 1) with Slow Internal Clock (**powerc**[10] = 1)	PLL Disabled[‡] During Standby	7.6 µs	$7.6\ \mu s + t_L{}^{\dagger\dagger}$
	PLL Enabled[‡] During Standby	—	7.6 µs
Hardware Stop (STOP Pin Asserted) or Software Stop (NOCK (**powerc**[9]) = 1)	PLL Disabled[‡] During Standby	$3T^\S$	$3T^\S + t_L{}^{\dagger\dagger}$
	PLL Enabled[‡] During Standby	—	$3T^\S$

† The PLL is deselected if the PLLSEL field (**pllc**[14]) is cleared, which is the default after reset. The PLL is selected if the PLLSEL field (**pllc**[14]) is set.

‡ The PLL is disabled (powered down) if the PLLEN field (**pllc**[15]) is cleared, which is the default after reset. The PLL is enabled (powered up) if the PLLEN field (**pllc**[15]) is set.

§ T = CLK clock cycle ($f_{CLK} = f_{CKI}$ if PLL deselected; $f_{CLK} = f_{CKI} \times M/2N$ if PLL enabled and selected).

†† t_L = PLL lock-in time (see Table 88 on page 139).

Timing Characteristics and Requirements (continued)

DSP Clock Generation
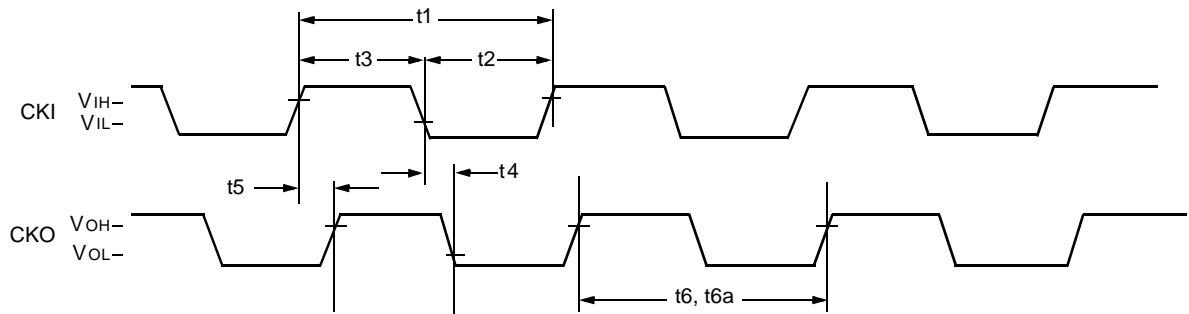
5-4009(F).d

**Figure 28. I/O Clock Timing Diagram**

Table 90. Timing Requirements for Input Clock

Abbreviated Reference	Parameter	Min	Max	Unit
t1	Clock In Period (high to high)	20	—[†]	ns
t2	Clock In Low Time (low to high)	10	—	ns
t3	Clock In High Time (high to low)	10	—	ns

† Device is fully static, t1 is tested at 100 ns input clock option, and memory hold time is tested at 0.1 s.

Table 91. Timing Characteristics for Input Clock and Output Clock

Abbreviated Reference	Parameter	Min	Max	Unit
t4	Clock Out High Delay (low to low)	—	10	ns
t5	Clock Out Low Delay (high to high)	—	10	ns
t6	Clock Out Period (high to high)	T[†]	—	ns
t6a	Clock Out Period with SLOWCLK Bit Set in **powerc** Register (high to high)	0.74	3.8	µs

† T = internal clock period (CLK).

Timing Characteristics and Requirements (continued)

Reset Circuit

The DSP16210 has two external reset pins: RSTB and TRST. At initial powerup, or if the supply voltage falls below VDD MIN[1] and a device reset is required, both TRST and RSTB must be asserted simultaneously to initialize the device.   Figure 29 shows two separate events:

1. Device reset at initial powerup.
2. Device reset following a drop in power supply.

Note: The TRST pin must be asserted even if the JTAG controller is not used by the application.
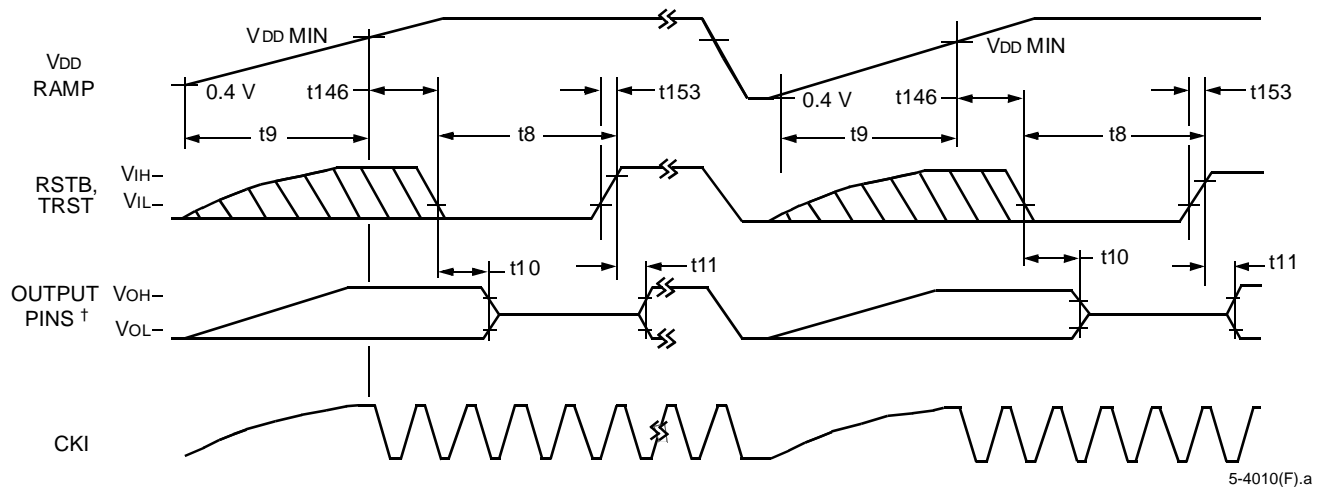
† When both INT0 and RSTB are asserted, all output and bidirectional pins (except TDO, which 3-states by JTAG control) are put in a
  3-state condition. With RSTB asserted and INT0 not asserted, EROM, ERAMHI, ERAMLO, ERAM, IO, and RWN outputs remain high, and
  CLK remains a free-running clock.

Figure 29. Powerup Reset and Device Reset Timing Diagram

Table 92. Timing Requirements for Powerup Reset and Device Reset

Abbreviated Reference	Parameter	Min	Max	Unit
t8	RSTB and TRST Reset Pulse (low to high)	7T†	—	ns
t9	VDD Ramp	—	10	ms
t146	VDD MIN to RSTB Low	2T†	—	ns
t153	RSTB and TRST Rise (low to high)	—	60	ns

† T = internal clock period (CLK).

Table 93. Timing Characteristics for Powerup Reset and Device Reset

Abbreviated Reference	Parameter	Min	Max	Unit
t10	RSTB Disable Time (low to 3-state)	—	100	ns
t11	RSTB Enable Time (high to valid)	—	100	ns

Note: The device needs to be clocked for at least six CKI cycles during reset after powerup. Otherwise, high currents flow.

1. See Table 83 on page 133, Recommended Operating Conditions.

Timing Characteristics and Requirements (continued)

Reset Synchronization
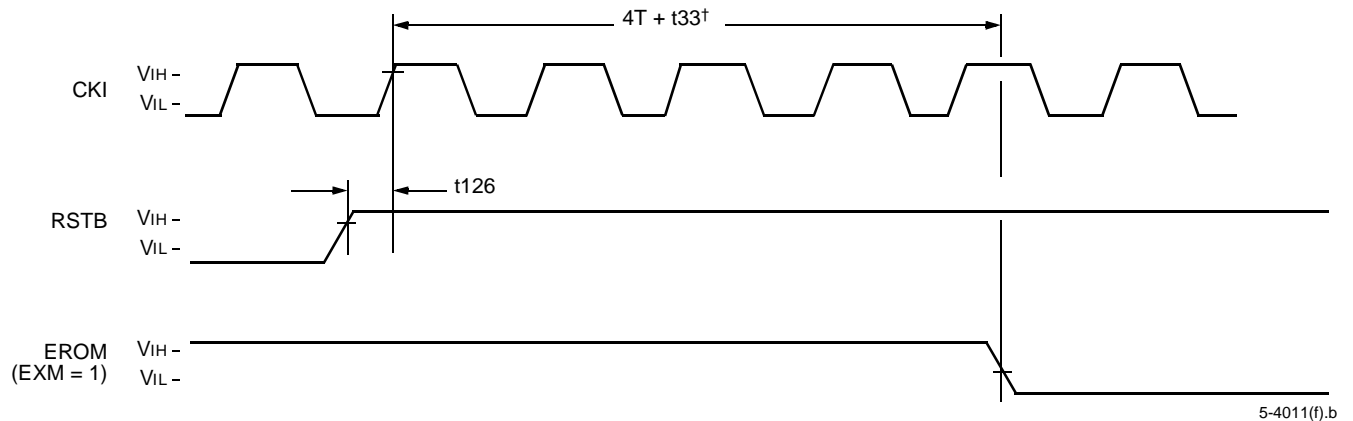
† See Table 101 on page 147 for EROM timing characteristics. T = internal clock period (CLK).

Figure 30. Reset Synchronization Timing

Table 94. Timing Requirements for Reset Synchronization Timing

Abbreviated Reference	Parameter	Min	Max	Unit
t126	Reset Setup (high to high)	3	$T/2 - 1$†	ns

† T = internal clock period (CLK).

Timing Characteristics and Requirements (continued)

JTAG
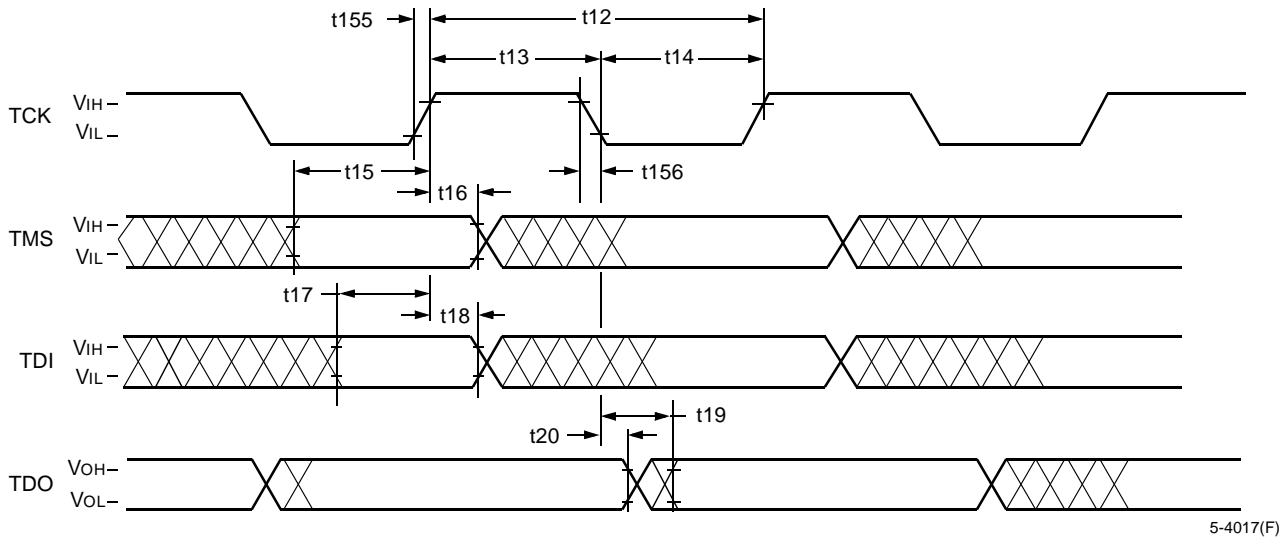
Figure 31. JTAG I/O Timing Diagram

Table 95. Timing Requirements for JTAG I/O

Abbreviated Reference	Parameter	Min	Max	Unit
t12	TCK Period (high to high)	50	—	ns
t13	TCK High Time (high to low)	22.5	—	ns
t14	TCK Low Time (low to high)	22.5	—	ns
t155	TCK Rise Transition Time (low to high)	0.6	—	V/ns
t156	TCK Fall Transition Time (high to low)	0.6	—	V/ns
t15	TMS Setup Time (valid to high)	7.5	—	ns
t16	TMS Hold Time (high to invalid)	5	—	ns
t17	TDI Setup Time (valid to high)	7.5	—	ns
t18	TDI Hold Time (high to invalid)	5	—	ns

Table 96. Timing Characteristics for JTAG I/O

Abbreviated Reference	Parameter	Min	Max	Unit
t19	TDO Delay (low to valid)	—	15	ns
t20	TDO Hold (low to invalid)	0	—	ns

Timing Characteristics and Requirements (continued)

Interrupt and Trap
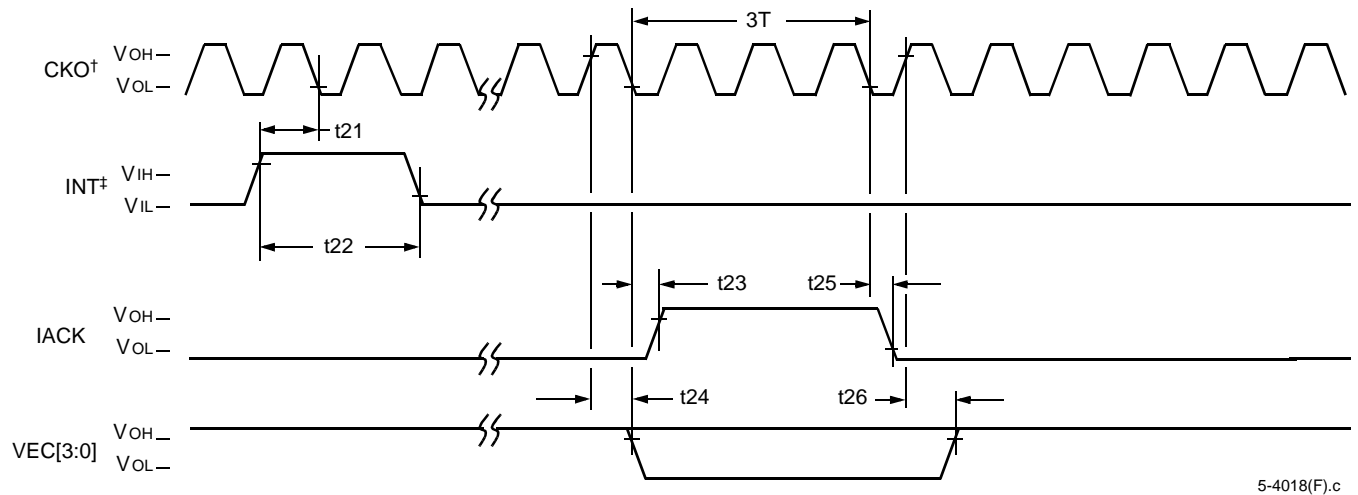
† CKO is free-running.
‡ INT is one of INT[3:0] or TRAP.

Figure 32. Interrupt and Trap Timing Diagram

Table 97. Timing Requirements for Interrupt and Trap

Note: Interrupt is asserted during an interruptible instruction and no other pending interrupts.

Abbreviated Reference	Parameter	Min	Max	Unit
t21	Interrupt Setup (high to low)	8	—	ns
t22	INT/TRAP Assertion Time (high to low)	2T†	—	ns

† T = internal clock period (CLK).

Table 98. Timing Characteristics for Interrupt and Trap

Note: Interrupt is asserted during an interruptible instruction and no other pending interrupts.

Abbreviated Reference	Parameter	Min	Max	Unit
t23	IACK Valid Time (low to high)	—	10	ns
t24	VEC Valid Time (high to low)	—	10	ns
t25	IACK Invalid Time (low to low)	—	10	ns
t26	VEC Invalid Time (high to high)	—	10	ns

Timing Characteristics and Requirements (continued)

Bit I/O
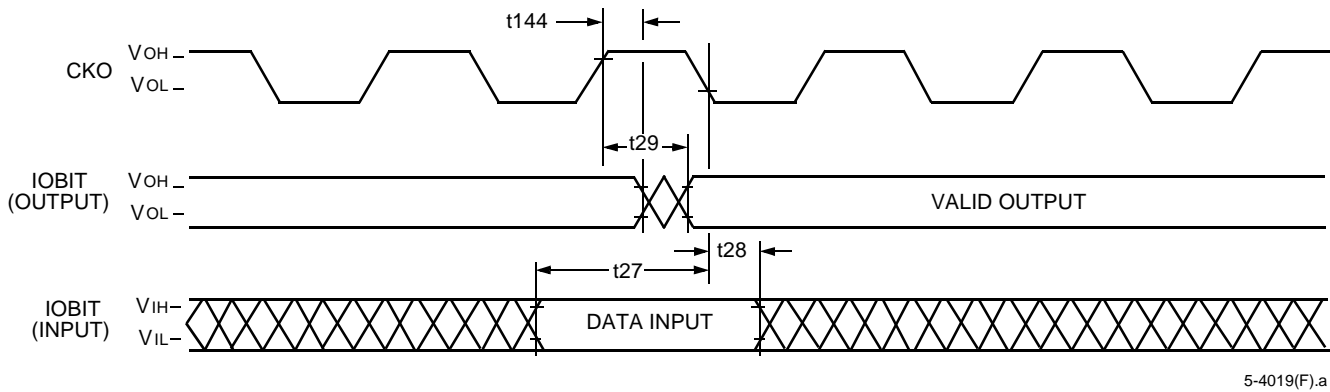
5-4019(F).a

Figure 33. Write Outputs Followed by Read Inputs (cbit = IMMEDIATE; a1 = sbit) Timing Characteristics

Table 99. Timing Requirements for BIO Input Read

Abbreviated Reference	Parameter	Min	Max	Unit
t27	IOBIT Input Setup Time (valid to low)	10	—	ns
t28	IOBIT Input Hold Time (low to invalid)	0	—	ns

Table 100. Timing Characteristics for BIO Output

Abbreviated Reference	Parameter	Min	Max	Unit
t29	IOBIT Output Valid Time (high to valid)	—	9	ns
t144	IOBIT Output Hold Time (high to invalid)	1	—	ns

Timing Characteristics and Requirements (continued)

External Memory Interface

The following timing diagrams, characteristics, and requirements do not apply to interactions with delayed external memory enables unless so stated. See the *DSP16210 Digital Signal Processor* Information Manual for a detailed description of the external memory interface including other functional diagrams. The term ENABLE refers to EROM, ERAM, IO, ERAMHI, and ERAMLO.
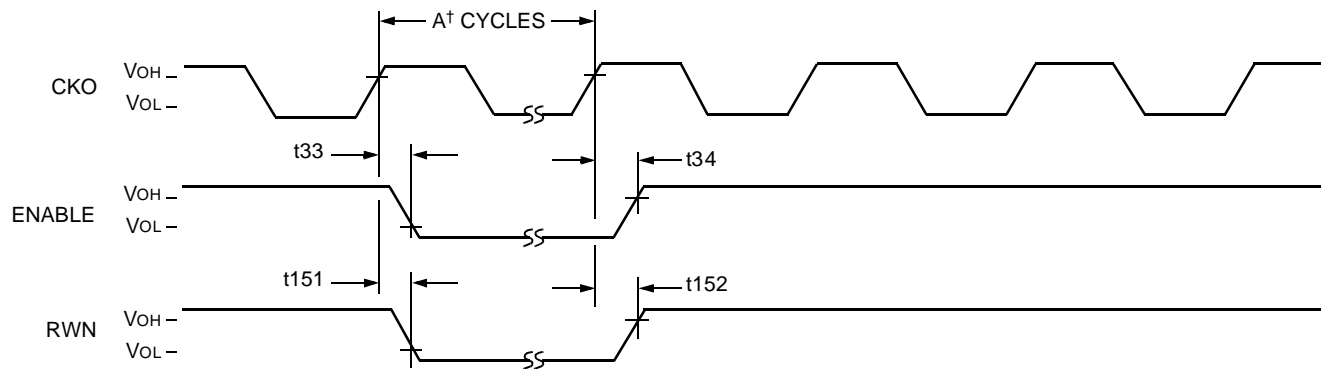
† A = number of DSP clock cycles programmed into the **mwait** register (XATIM, YATIM, IATIM) for the access.

Figure 34. Enable Transition Timing

Table 101. Timing Characteristics for Memory Enables and RWN

Abbreviated Reference	Parameter	Condition	Min	Max	Unit
t33	CKO to ENABLE Active (high to low)	DENB† = 0	0	4.5	ns
		DENB† = 1	$T/2$‡	$T/2$‡ + 7	ns
t34	CKO to ENABLE Inactive (high to high)	DENB† = 0	−1	4	ns
		DENB† = 1	$T/2$‡ − 1	$T/2$‡ + 6	ns
t151	CKO to RWN Active (high to low)	RWNADV§ = 1	0	4.5	ns
t152	CKO to RWN Inactive (high to high)	—	−1	4	ns

† DENB is replaced with the DENB[2:0] bit of the **ioc** register that corresponds to the memory segment that is accessed. DENB is DENB2 for the IO segment, DENB1 for the ERAMHI and ERAMLO segments, and DENB0 for the EROM segment.
‡ T = internal clock period (CLK).
§ RWNADV is bit 3 of the **ioc** register.

Timing Characteristics and Requirements (continued)

External Memory Interface (continued)
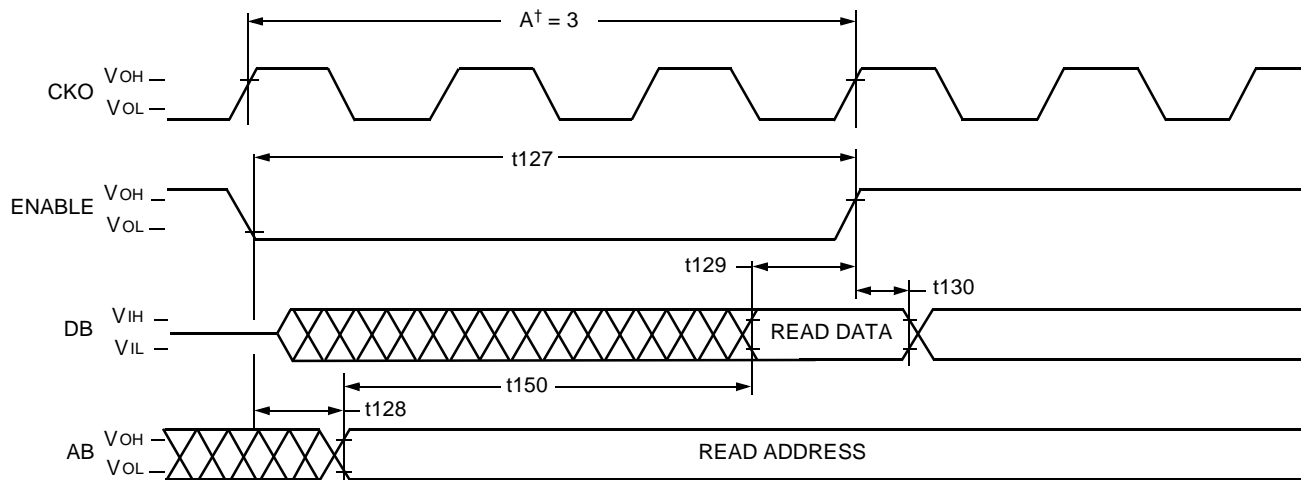
5-4021(f).d

† A = number of DSP clock cycles programmed into the **mwait** register (XATIM, YATIM, IATIM) for the access.

Figure 35. External Memory Data Read Timing Diagram (No Delayed Enable)

Table 102. Timing Characteristics for External Memory Access (DENB† = 0)

Abbreviated Reference	Parameter	Min	Max	Unit
t127	Enable Width (low to high)	$(T \times A) - 2$‡	—	ns
t128	Address Valid (enable low to valid)	—	2	ns

† DENB is replaced with the DENB[2:0] bit of the **ioc** register that corresponds to the memory segment that is accessed. DENB is DENB2 for the IO segment, DENB1 for the ERAMHI and ERAMLO segments, and DENB0 for the EROM segment.
‡ T = internal clock period (CLK).

Table 103. Timing Requirements for External Memory Read (DENB† = 0)

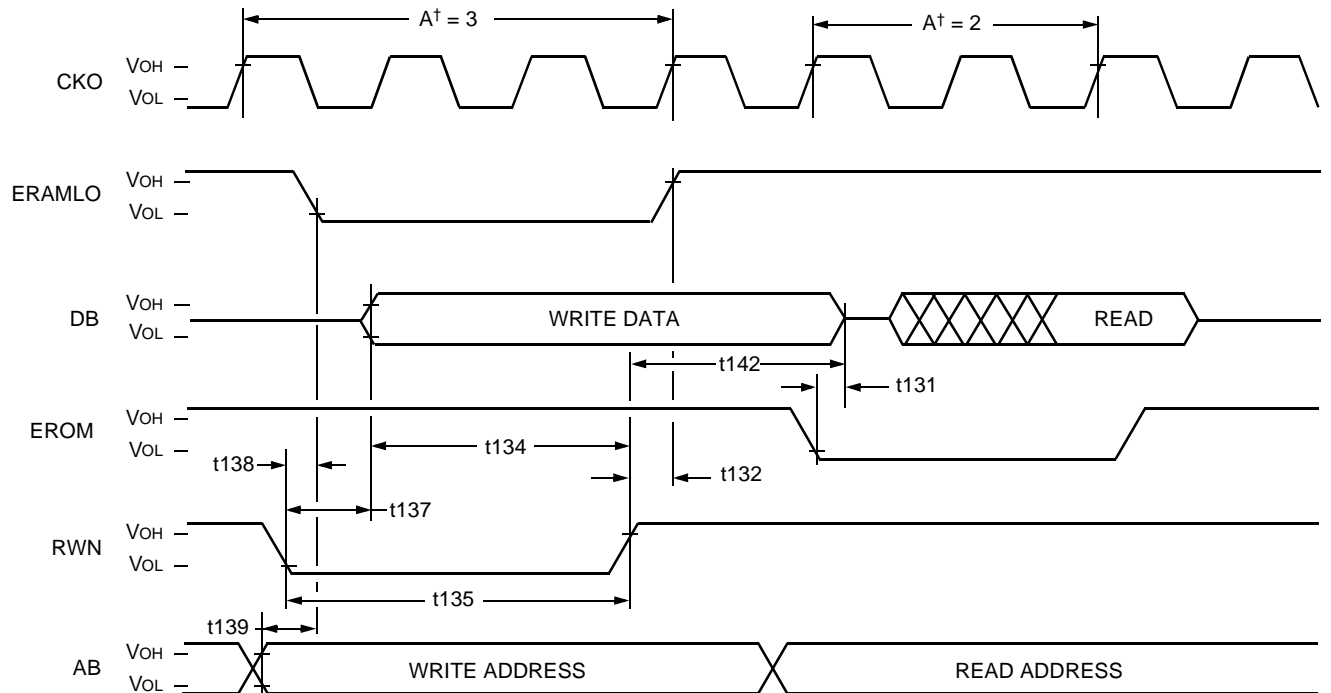| Abbreviated Reference	Parameter	Min	Max	Unit
t129	Read Data Setup (valid to enable high)	8.5	—	ns
t130	Read Data Hold (enable high to hold)	0	—	ns
t150	External Memory Access Time (valid to valid)	—	$(T \times A) - 10$‡	ns

† DENB is replaced with the DENB[2:0] bit of the **ioc** register that corresponds to the memory segment that is accessed. DENB is DENB2 for the IO segment, DENB1 for the ERAMHI and ERAMLO segments, and DENB0 for the EROM segment.
‡ T = internal clock period (CLK).

Timing Characteristics and Requirements (continued)

External Memory Interface (continued)
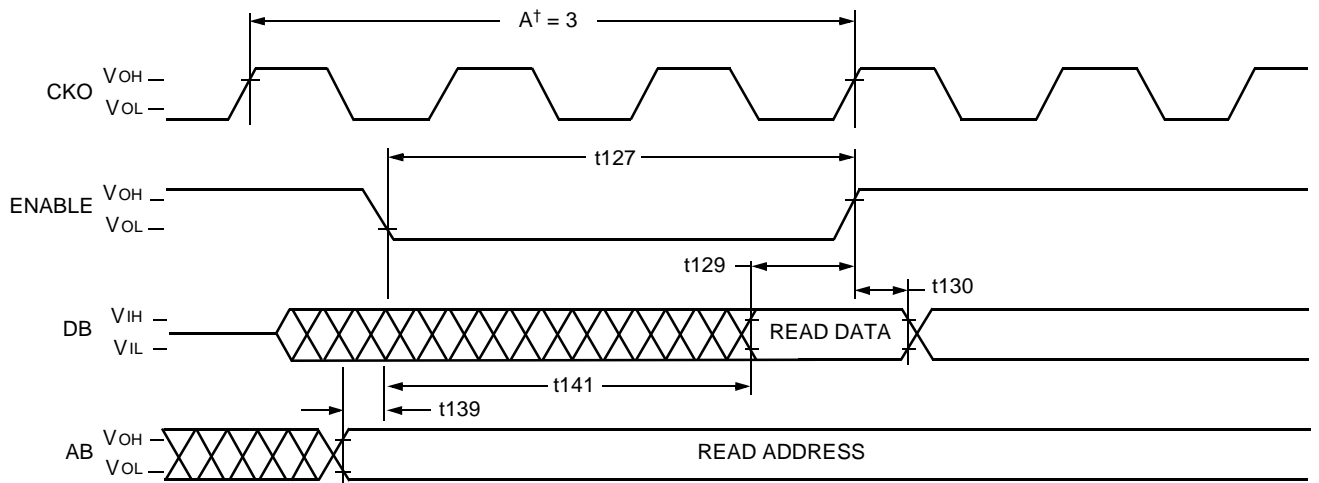
5-4021(f).e

† A = number of DSP clock cycles programmed into the **mwait** register (XATIM, YATIM, IATIM) for the access.

Figure 36. External Memory Data Read Timing Diagram (Delayed Enable)

Table 104. Timing Characteristics for External Memory Access (DENB† = 1)

Abbreviated Reference	Parameter	Min	Max	Unit
t127	Enable Width (low to high)	$(T \times (A - 0.5)) - 2$‡	—	ns
t139	Address Valid (valid to enable low)	$T/2 - 3$‡	—	ns

† DENB is replaced with the DENB[2:0] bit of the **ioc** register that corresponds to the memory segment that is accessed. DENB is DENB2 for the IO segment, DENB1 for the ERAMHI and ERAMLO segments, and DENB0 for the EROM segment.
‡ T = internal clock period (CLK).

Table 105. Timing Requirements for External Memory Read (DENB† = 1)

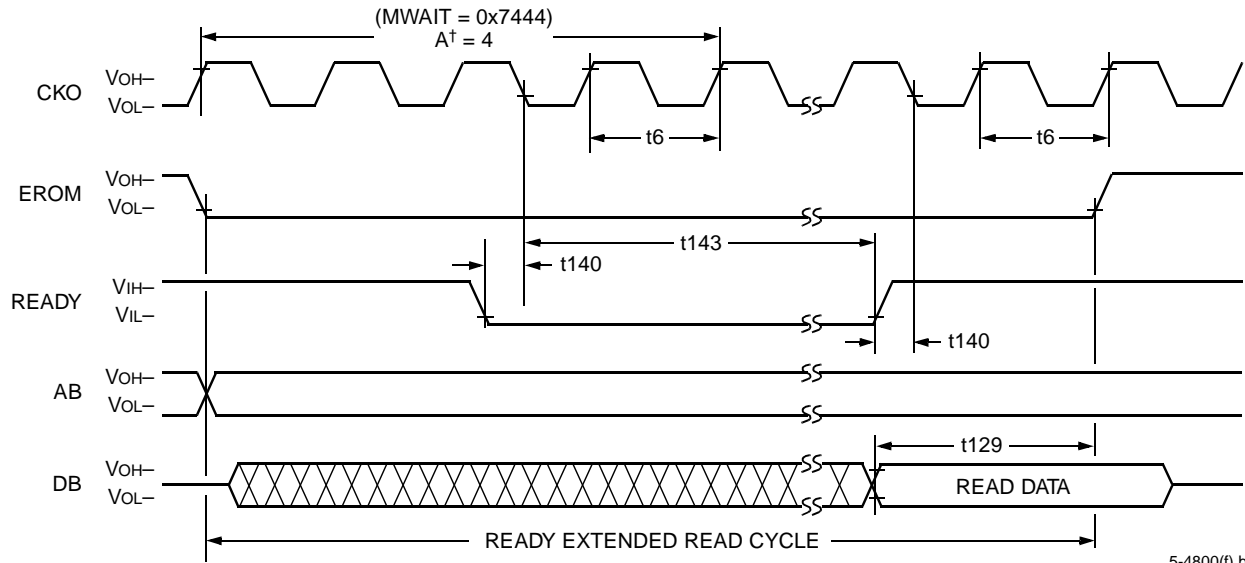| Abbreviated Reference	Parameter	Min	Max	Unit
t129	Read Data Setup (valid to enable high)	8.5	—	ns
t130	Read Data Hold (enable high to hold)	0	—	ns
t141	External Memory Access Time (enable low to valid)	—	$(T \times (A - 0.5)) - 11$‡	ns

† DENB is replaced with the DENB[2:0] bit of the **ioc** register that corresponds to the memory segment that is accessed. DENB is DENB2 for the IO segment, DENB1 for the ERAMHI and ERAMLO segments, and DENB0 for the EROM segment.
‡ T = internal clock period (CLK).

Timing Characteristics and Requirements (continued)

External Memory Interface (continued)
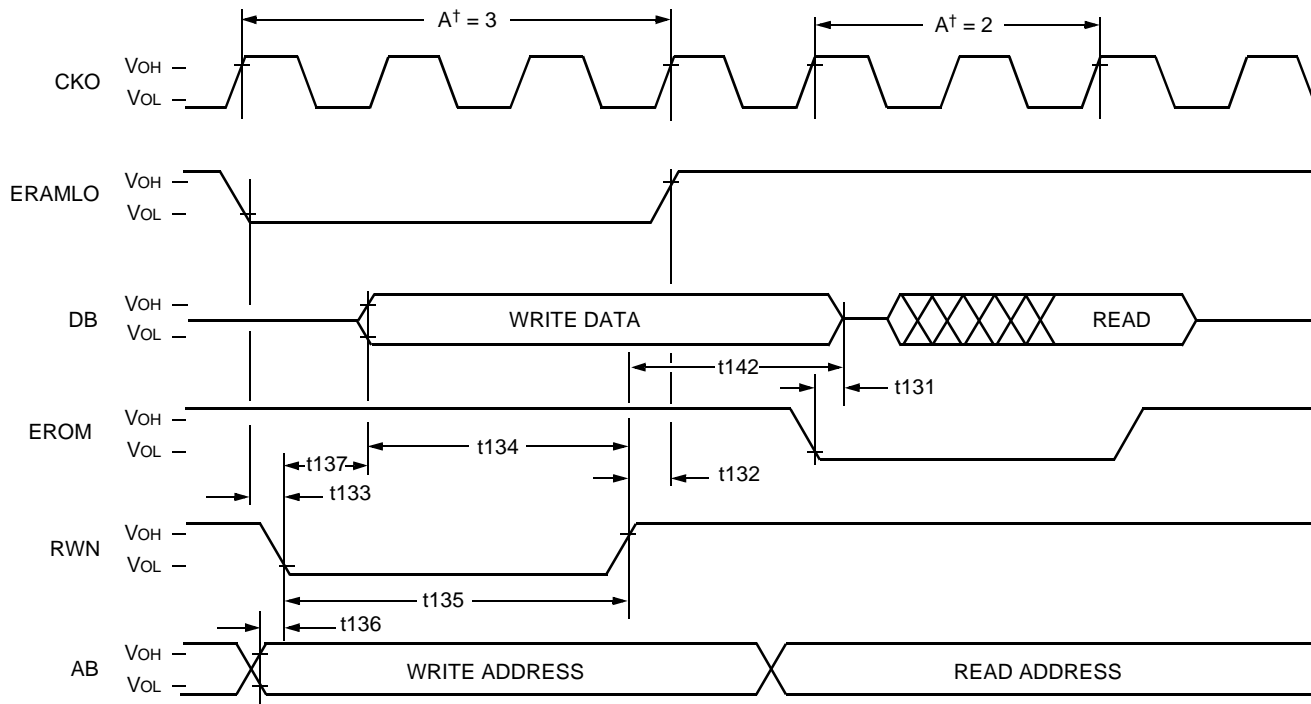
† A = number of DSP clock cycles programmed into the **mwait** register (XATIM, YATIM, IATIM) for the access. In the example depicted in this drawing, YATIM = 3 and XATIM = 2.

Figure 37. External Memory Data Write Timing Diagram (DENB2 = 0, DENB1 = 0, DENB0 = 0)

Table 106. Timing Characteristics for External Memory Data Write (RWNADV† = 0, DENB‡ = 0)

Abbreviated Reference	Parameter	Condition	Min	Max	Unit
t131	Write Overlap (enable low to 3-state)	—	—	0	ns
t132	RWN Advance (RWN high to enable high)	—	0	2	ns
t133	RWN Delay (enable low to RWN low)	—	0	3	ns
t134	Write Data Setup (data valid to RWN high)	WDDLY§ = 0	$(T \times (A - 0.5)) - 3$††	—	ns
		WDDLY§ = 1	$(T \times (A - 1)) - 3$††	—	ns
t135	RWN Width (low to high)	—	$(T \times A) - 3$††	—	ns
t136	Write Address Setup (address valid to RWN low)	—	0	—	ns
t137	Write Data Activation Delay (RWN low to DB active)	WDDLY§ = 0	$T/2 - 4$††	—	ns
		WDDLY§ = 1	$T - 4$††	—	ns
t142	Write Data Deactivation Delay (RWN high to DB 3-state)	WDDLY§ = 0	—	$T/2$	ns
		WDDLY§ = 1	—	T	ns

† RWNADV is bit 3 of the **ioc** register.
‡ DENB is replaced with the DENB[2:0] bit of the **ioc** register that corresponds to the memory segment that is accessed. DENB is DENB2 for the IO segment, DENB1 for the ERAMHI and ERAMLO segments, and DENB0 for the EROM segment.
§ WDDLY is bit 10 of the **ioc** register.
†† T = internal clock period (CLK).

Timing Characteristics and Requirements (continued)

External Memory Interface (continued)

† A = number of DSP clock cycles programmed into the **mwait** register (XATIM, YATIM, IATIM) for the access.

Figure 38. External Memory Data Write Timing Diagram (DENB2 = 0, DENB1 = 1, DENB0 = 0)

Table 107. Timing Characteristics for External Memory Data Write (RWNADV† = 1, DENB‡ = 1)

Abbreviated Reference	Parameter	Condition	Min	Max	Unit
t131	Write Overlap (enable low to 3-state)	—	—	0	ns
t132	RWN Advance (RWN high to enable high)	—	0	2	ns
t134	Write Data Setup (data valid to RWN high)	WDDLY§ = 0	$(T \times (A - 0.5)) - 3$††	—	ns
		WDDLY§ = 1	$(T \times (A - 1)) - 3$††	—	ns
t135	RWN Width (low to high)	—	$(T \times A) - 3$††	—	ns
t137	Write Data Activation Delay (RWN low to DB active)	WDDLY§ = 0	$T/2 - 4$††	—	ns
		WDDLY§ = 1	$T - 4$††	—	ns
t138	Enable Delay (RWN low to enable Low)	—	$T/2 - 3$††	—	ns
t139	Address Valid (valid to enable low)	—	$T/2 - 3$††	—	ns
t142	Write Data Deactivation Delay (RWN high to DB 3-state)	WDDLY§ = 0	—	$T/2$	ns
		WDDLY§ = 1	—	T	ns

† RWNADV is bit 3 of the **ioc** register.
‡ DENB is replaced with the DENB[2:0] bit of the **ioc** register that corresponds to the memory segment that is accessed. DENB is DENB2 for the IO segment, DENB1 for the ERAMHI and ERAMLO segments, and DENB0 for the EROM segment.
§ WDDLY is bit 10 of the **ioc** register.
†† T = internal clock period (CLK).

Timing Characteristics and Requirements (continued)

External Memory Interface (continued)

† A = number of DSP clock cycles programmed into the **mwait** register (XATIM, YATIM, IATIM) for the access.

Figure 39. READY Extended Read Cycle Timing

Table 108. Timing Requirements for READY Extended Read Cycle Timing

Abbreviated Reference	Parameter	Min	Max	Unit
t6	Clock Out Period (high to high)	T†	—	ns
t140	READY Setup (valid to CKO low)	6	—	ns
t143	READY Hold (CKO low to invalid)	6	—	ns
t129	Read Data Setup (valid to enable high)	8.5	—	ns

† T = internal clock period (CLK).

Timing Characteristics and Requirements (continued)

PHIF16

For the PHIF16, READ means read by the external user (output by the DSP); WRITE is similarly defined. In the 8-bit external bus configuration, 8-bit reads/writes are identical to one-half of a 16-bit access. In the 16-bit external bus mode, accesses are identical to 8-bit accesses in the 8-bit external bus mode.
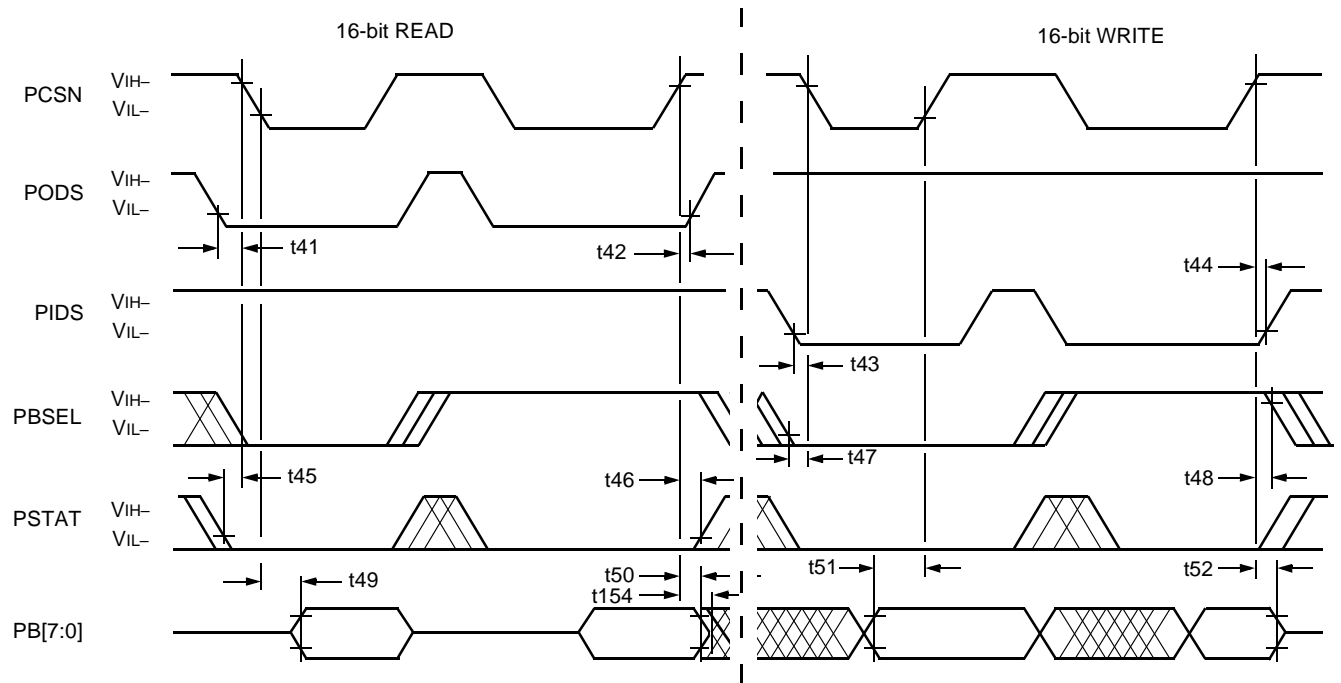
5-4036(F)

Note: This timing diagram shows accesses using the PCSN signal to initiate and complete a transaction. The transactions can also be initiated and completed with the PIDS and PODS signals. An output transaction (read) is initiated by PCSN or PODS going low, whichever comes last. An output transaction is completed by PCSN or PODS going high, whichever comes first. An input transaction is initiated by PCSN or PIDS going low, whichever comes last. An input transaction is completed by PCSN or PIDS going high, whichever comes first.

Figure 40. PHIF16 *Intel* Mode Signaling (Read and Write) Timing Diagram

Table 109. Timing Requirements for PHIF16 *Intel* Mode Signaling (Read and Write)

Abbreviated Reference	Parameter	Min	Max	Unit
t41	PODS to PCSN Setup (low to low)	0	—	ns
t42	PCSN to PODS Hold (high to high)	0	—	ns
t43	PIDS to PCSN Setup (low to low)	0	—	ns
t44	PCSN to PIDS Hold (high to high)	0	—	ns
t45[†]	PSTAT to PCSN Setup (valid to low)	4	—	ns
t46[†]	PCSN to PSTAT Hold (high to invalid)	0	—	ns
t47[†]	PBSEL to PCSN Setup (valid to low)	6	—	ns
t48[†]	PCSN to PBSEL Hold (high to invalid)	0	—	ns
t51[†]	PB Write to PCSN Setup (valid to high)	10	—	ns
t52[†]	PCSN to PB Write Hold (high to invalid)	4	—	ns

† If PIDS or PODS is the controlling signal instead of PCSN, then all requirements that reference PCSN apply instead to PIDS or PODS.

Timing Characteristics and Requirements (continued)

PHIF16 (continued)

Table 110. Timing Characteristics for PHIF16 *Intel* Mode Signaling (Read and Write)

Abbreviated Reference	Parameter	Min	Max	Unit
t49[†]	PCSN to PB Read (low to valid)	—	12	ns
t50[†]	PCSN to PB Read Hold (high to invalid)	1	—	ns
t154	PCSN to PB Read 3-state (high to 3-state)	—	8	ns

† If PIDS or PODS is the controlling signal instead of PCSN, then all characteristics that reference PCSN apply instead to PIDS or PODS.

Timing Characteristics and Requirements (continued)

PHIF16 (continued)

**Note:** This diagram assumes an 8-bit external interface.
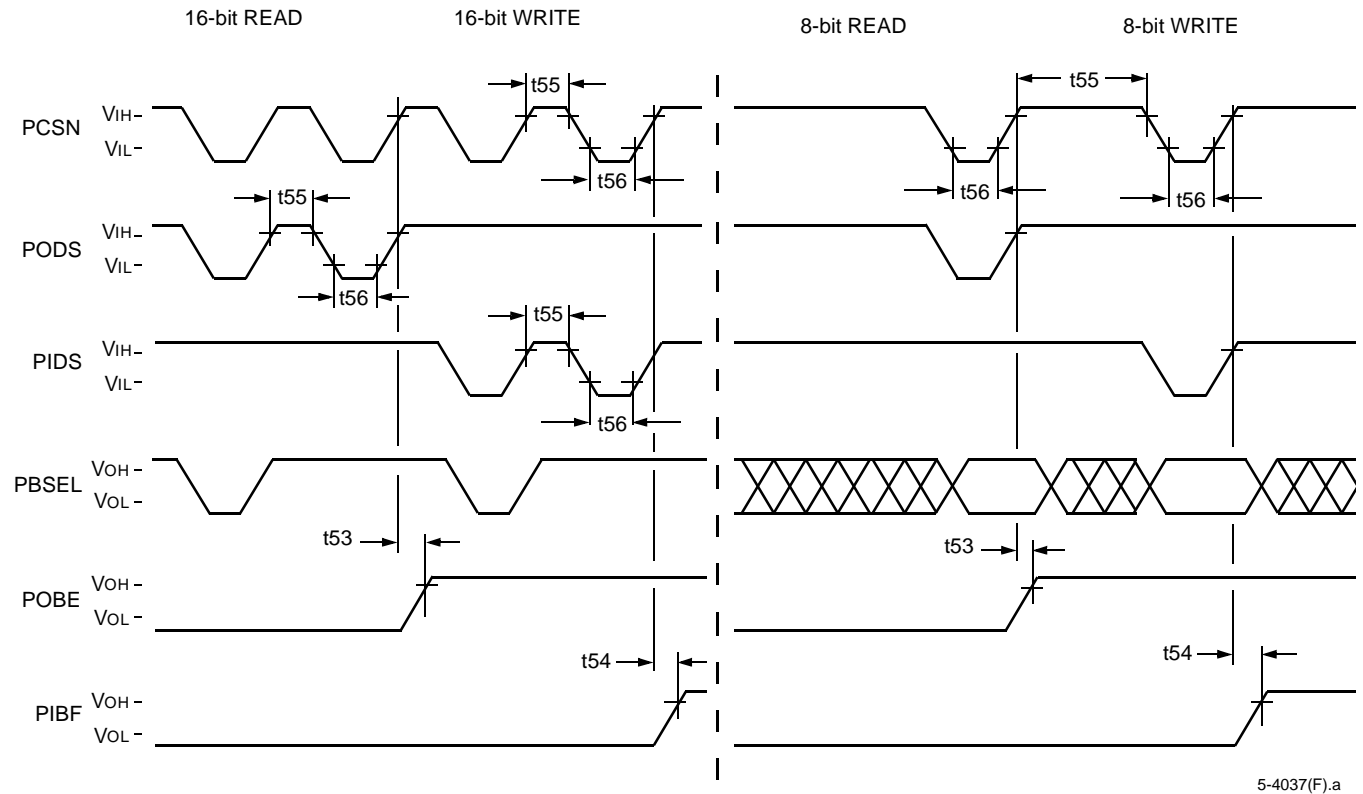
Figure 41. PHIF16 *Intel* Mode Signaling (Pulse Period and Flags) Timing Diagram

Table 111. Timing Requirements for PHIF16 *Intel* Mode Signaling (Pulse Period and Flags)

Abbreviated Reference	Parameter	Min	Max	Unit
t55	PCSN/PODS/PIDS Pulse Width (high to low)	20.5	—	ns
t56	PCSN/PODS/PIDS Pulse Width (low to high)	20.5	—	ns

Table 112. Timing Characteristics for PHIF16 *Intel* Mode Signaling (Pulse Period and Flags)

Abbreviated Reference	Parameter	Min	Max	Unit
t53[†]	PCSN/PODS to POBE[‡] (high to high)	—	15	ns
t54[†]	PCSN/PIDS to PIBF[‡](high to high)	—	15	ns

† t53 should be referenced to the rising edge of PCSN or PODS, whichever comes first. t54 should be referenced to the rising edge of PCSN or PIDS, whichever comes first.

‡ POBE and PIBF can be programmed to be the opposite logic levels shown in the diagram (positive assertion levels shown). t53 and t54 apply to the inverted levels as well as those shown.

Timing Characteristics and Requirements (continued)

PHIF16 (continued)

16-bit READ   16-bit WRITE
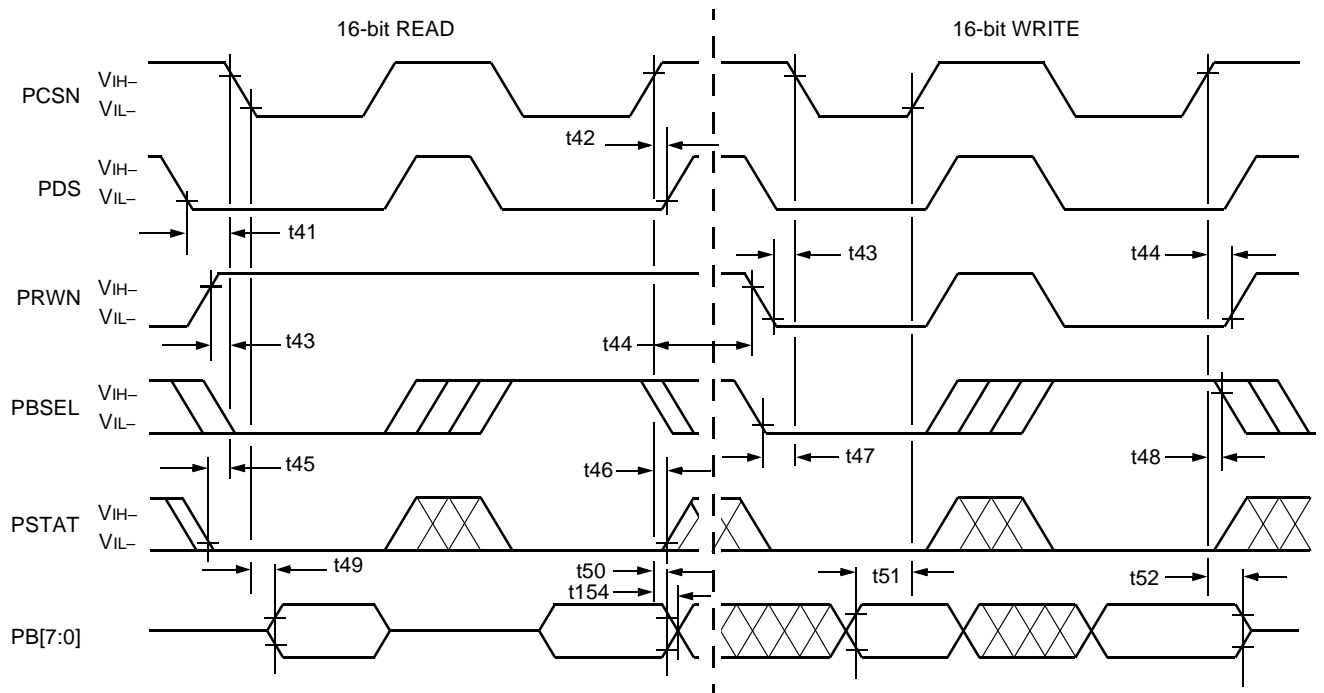
5-4038(F).a

Note: This timing diagram shows accesses using the PCSN signal to initiate and complete a transaction. The transactions can also be initiated and completed with the PIDS and PODS signals. An output transaction (read) is initiated by PCSN or PODS going low, whichever comes last. An output transaction is completed by PCSN or PODS going high, whichever comes first. An input transaction is initiated by PCSN or PIDS going low, whichever comes last. An input transaction is completed by PCSN or PIDS going high, whichever comes first.

Figure 42. PHIF16 *Motorola* Mode Signaling (Read and Write) Timing Diagram

Table 113. Timing Requirements for PHIF16 *Motorola* Mode Signaling (Read and Write)

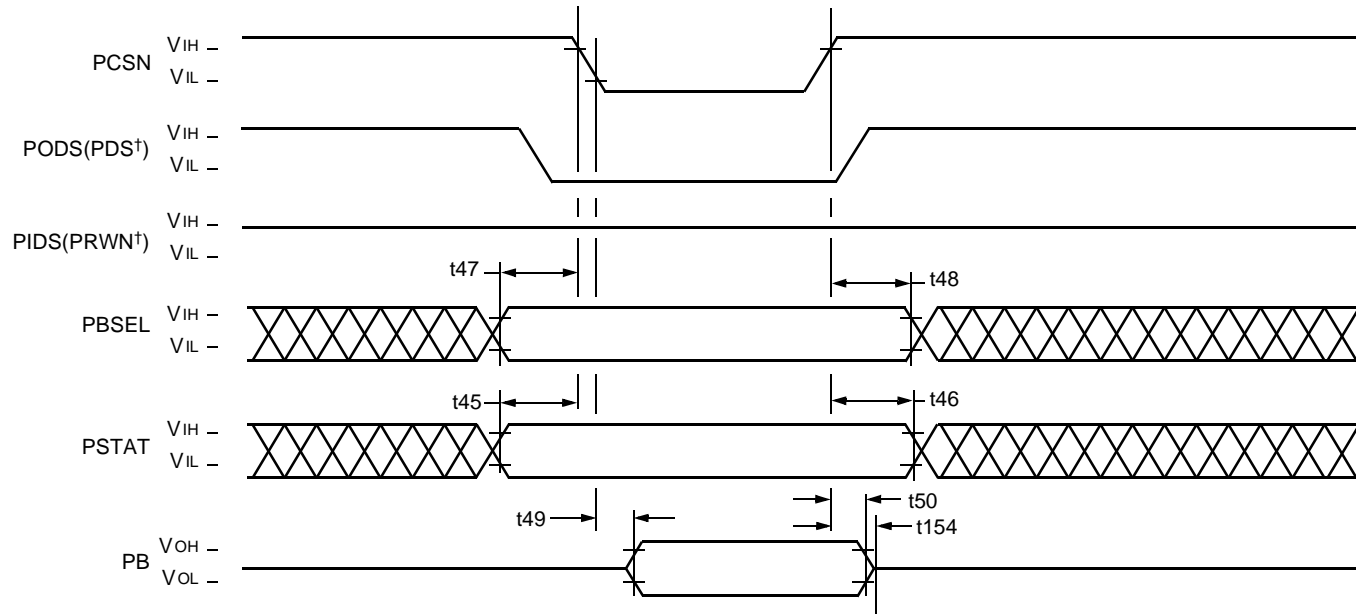| Abbreviated Reference	Parameter	Min	Max	Unit
t41	PDS[†] to PCSN Setup (valid to low)	0	—	ns
t42	PCSN to PDS[†] Hold (high to invalid)	0	—	ns
t43	PRWN to PCSN Setup (valid to low)	4	—	ns
t44	PCSN to PRWN Hold (high to invalid)	0	—	ns
t45[†]	PSTAT to PCSN Setup (valid to low)	4	—	ns
t46[†]	PCSN to PSTAT Hold (high to invalid)	0	—	ns
t47[†]	PBSEL to PCSN Setup (valid to low)	6	—	ns
t48[†]	PCSN to PBSEL Hold (high to invalid)	0	—	ns
t51[†]	PB Write to PCSN Setup (valid to high)	10	—	ns
t52[†]	PCSN to PB Write Hold (high to invalid)	4	—	ns

† PDS is programmable to be active-high or active-low. It is shown active-low in Figure 42. POBE and PIBF may be programmed to be the opposite logic levels shown in the diagram. t53 and t54 apply to the inverted levels as well as those shown.

Timing Characteristics and Requirements (continued)

PHIF16 (continued)

Table 114. Timing Characteristics for PHIF16 *Motorola* Mode Signaling (Read and Write)
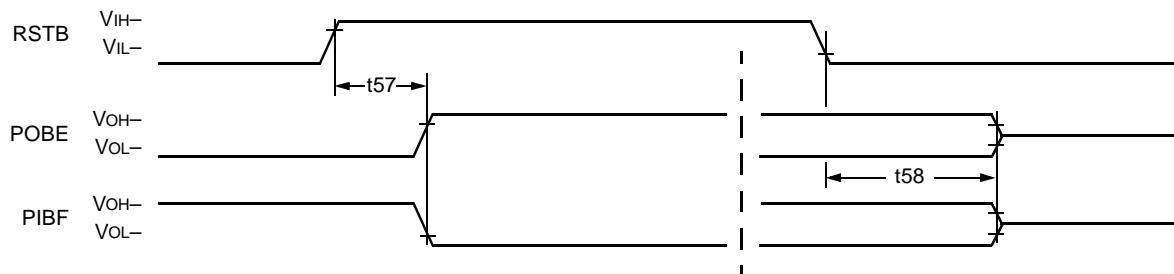
Abbreviated Reference	Parameter	Min	Max	Unit
t49[†]	PCSN to PB Read (low to valid)	—	12	ns
t50[†]	PCSN to PB Read (high to invalid)	0	—	ns
t154	PCSN to PB Read 3-state (high to 3-state)	—	8	ns

† If PIDS or PODS is the controlling signal instead of PCSN, then all characteristics that reference PCSN apply instead to PIDS or PODS.

Timing Characteristics and Requirements (continued)

PHIF16 (continued)

Note: This diagram assumes an 8-bit external interface.

5-4039(F).a
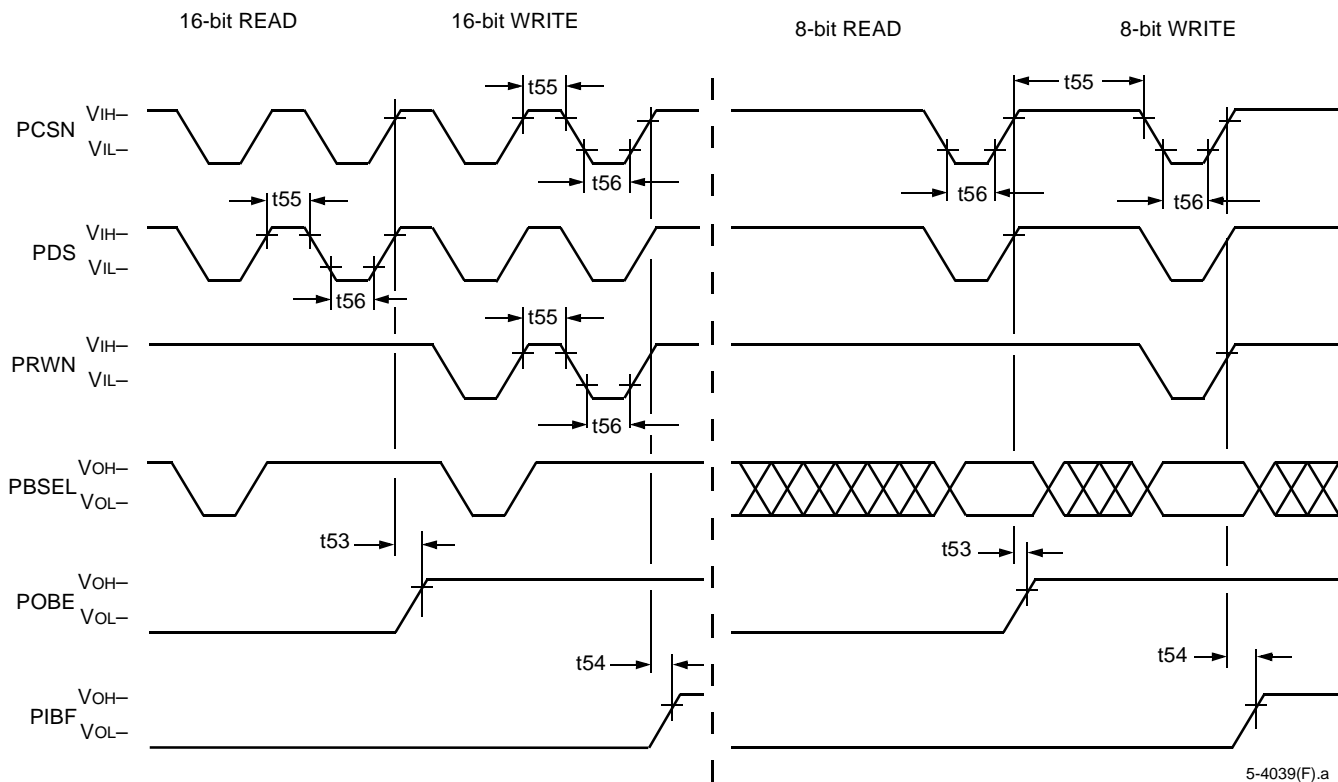
Figure 43. PHIF16 *Motorola* Mode Signaling (Pulse Period and Flags) Timing Diagram

Table 115. Timing Characteristics for PHIF16 *Motorola* Mode Signaling (Pulse Period and Flags)

Abbreviated Reference	Parameter	Min	Max	Unit
t53[†]	PCSN/PDS[‡] to POBE[‡] (high to high)	—	15	ns
t54[†]	PCSN/PDS[‡] to PIBF[‡] (high to high)	—	15	ns

† An input/output transaction is initiated by PCSN or PDS going low, whichever comes last. For example, t53 and t54 should be referenced to PDS going low, if PDS goes low after PCSN. An input/output transaction is completed by PCSN or PDS going high, whichever comes first. All requirements referenced to PCSN should be referenced to PDS, if PDS is the controlling signal. PRWN should never be used to initiate or complete a transaction.

‡ PDS is programmable to be active-high or active-low. It is shown active-low in Figure 43. POBE and PIBF may be programmed to be the opposite logic levels shown in the diagram. t53 and t54 apply to the inverted levels as well as those shown.

Table 116. Timing Requirements for PHIF16 *Motorola* Mode Signaling (Pulse Period and Flags)

Abbreviated Reference	Parameter	Min	Max	Unit
t55	PCSN/PDS/PRWN Pulse Width (high to low)	20	—	ns
t56	PCSN/PDS/PRWN Pulse Width (low to high)	20	—	ns

Timing Characteristics and Requirements (continued)

PHIF16 (continued)

† *Motorola* mode signal name.

5-4040(F).a

Figure 44. PHIF16 *Intel* or *Motorola* Mode Signaling (Status Register Read) Timing Diagram

Table 117. Timing Requirements for *Intel* and *Motorola* Mode Signaling (Status Register Read)

Abbreviated Reference	Parameter	Min	Max	Unit
t45†	PSTAT to PCSN Setup (valid to low)	4	—	ns
t46‡	PCSN to PSTAT Hold (high to invalid)	0	—	ns
t47†	PBSEL to PCSN Setup (valid to low)	6	—	ns
t48‡	PCSN to PBSEL Hold (high to invalid)	0	—	ns

† t45 and t47 are referenced to the falling edge of PCSN or PODS(PDS), whichever occurs last.
‡ t46 and t48 are referenced to the rising edge of PCSN or PODS(PDS), whichever occurs first.

Table 118. Timing Characteristics for *Intel* and *Motorola* Mode Signaling (Status Register Read)

Abbreviated Reference	Parameter	Min	Max	Unit
t49†	PCSN to PB Read (low to valid)	—	12	ns
t50‡	PCSN to PB Read Hold (high to invalid)	0	—	ns
t154‡	PCSN to PB Read 3-state (high to 3-state)	—	8	ns

† t49 is referenced to the falling edge of PCSN or PODS(PDS), whichever occurs last.
‡ t50 and t154 are referenced to the rising edge of PCSN or PODS(PDS), whichever occurs first.

Timing Characteristics and Requirements (continued)

PHIF16 (continued)

5-4775 (F)

Figure 45. PIBF and POBE Reset Timing Diagram

Table 119. PHIF16 Timing Characteristics for PIBF and POBE Reset

Abbreviated Reference	Parameter	Min	Max	Unit
t57	RSTB Disable to POBE/PIBF[†] (high to valid)	—	35	ns
t58	RSTB Enable to POBE/PIBF[†] (low to invalid)	3	35	ns

† After reset, POBE and PIBF always go to the levels shown, indicating output buffer empty and input buffer full. The DSP program, however, may later invert the definition of the logic levels for POBE and PIBF. t57 and t58 continue to apply.
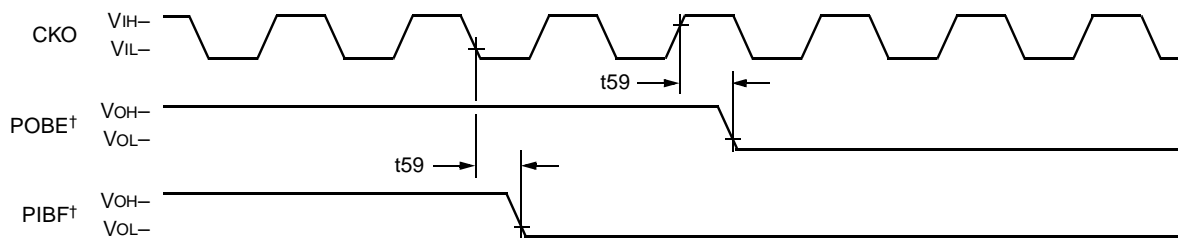
5-4776 (F)

† POBE and PIBF can be programmed to be active-high or active-low. They are shown active-high. The timing characteristic for active-low is the same as for active-high.

Figure 46. POBE and PIBF Disable Timing Diagram

Table 120. PHIF16 Timing Characteristics for POBE and PIBF Disable

Abbreviated Reference	Parameter	Min	Max	Unit
t59	CKO to POBE/PIBF Disable (high/low to disable)	—	15	ns

Timing Characteristics and Requirements (continued)

Simple Serial I/O
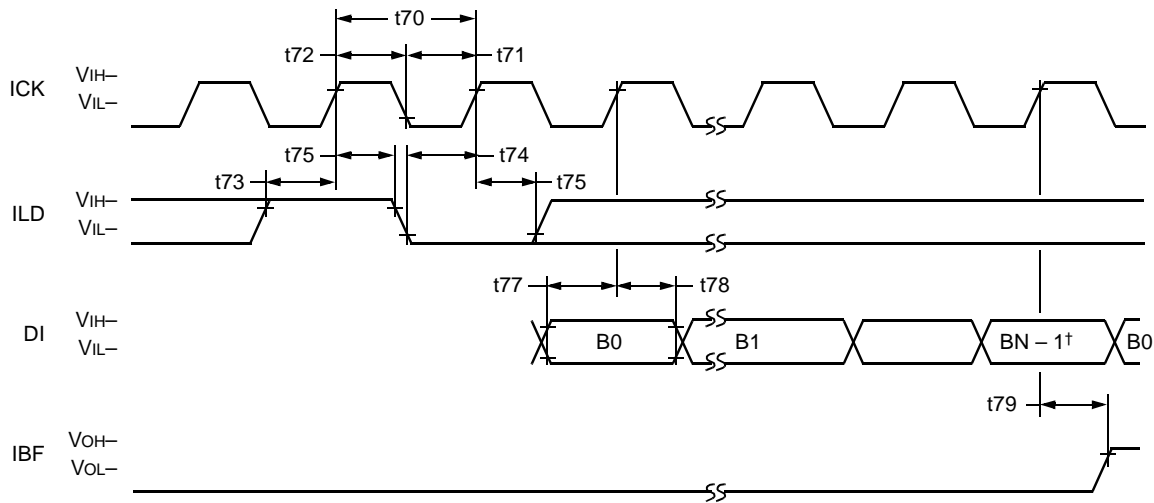
† N = 8 or 16 bits.

5-4777 (F)

Figure 47. SSIO Passive Mode Input Timing Diagram

Table 121. Timing Requirements for Serial Inputs (Passive Mode)

| Abbreviated Reference	Parameter	Min	Max	Unit
t70	Clock Period (high to high)	38	—†	ns
t71	Clock Low Time (low to high)	16	—	ns
t72	Clock High Time (high to low)	16	—	ns
t73	Load High Setup (high to high)	8	—	ns
t74	Load Low Setup (low to high)	8	—	ns
t75	Load High Hold (high to invalid)	0	—	ns
t77	Data Setup (valid to high)	7	—	ns
t78	Data Hold (high to invalid)	0	—	ns

† Device is fully static; t70 is tested at 200 ns.

Table 122. Timing Characteristics for Serial Outputs (Passive Mode)

Abbreviated Reference	Parameter	Min	Max	Unit
t79	IBF Delay (high to high)	—	15	ns

Timing Characteristics and Requirements (continued)

Simple Serial I/O (continued)
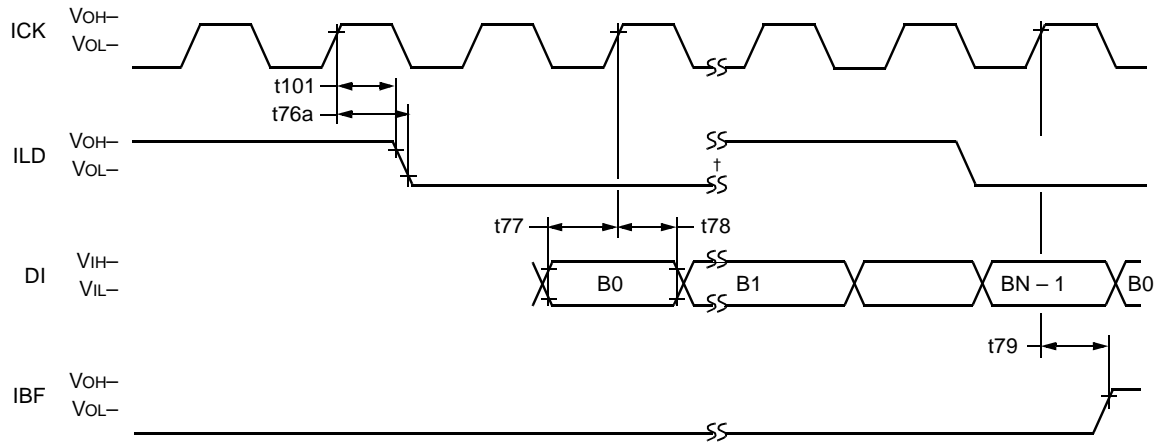
† ILD goes high during bit 6 (of 0:15), N = 8 or 16 bits.

Figure 48. SSIO Active Mode Input Timing Diagram

Table 123. Timing Requirements for Serial Inputs (Active Mode)

Abbreviated Reference	Parameter	Min	Max	Unit
t77	Data Setup (valid to high)	8	—	ns
t78	Data Hold (high to invalid)	0	—	ns

Table 124. Timing Characteristics for Serial Outputs (Active Mode)

Abbreviated Reference	Parameter	Min	Max	Unit
t76a	ILD Delay (high to low)	—	20	ns
t101	ILD Hold (high to invalid)	1	—	ns
t79	IBF Delay (high to high)	—	20	ns

Timing Characteristics and Requirements (continued)

Simple Serial I/O (continued)
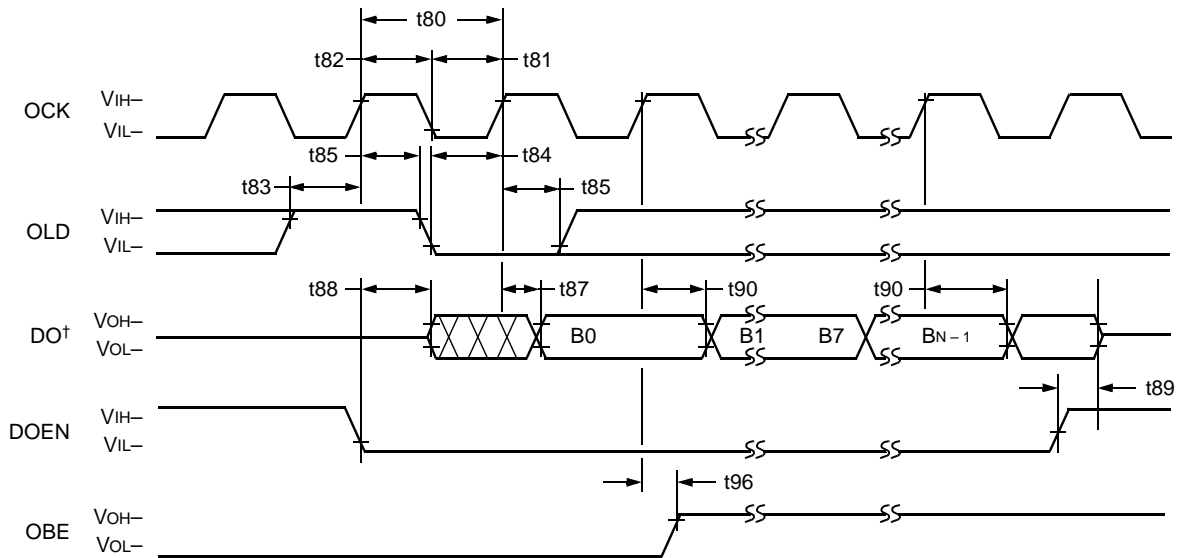
† See **SSIOC** register, MSB field, to determine if B0 is the MSB or LSB. See **SSIOC** register, ILEN field, to determine if the DO word length is 8 bits or 16 bits. It is assumed that the DODLY bit is 0.

Figure 49. SSIO Passive Mode Output Timing Diagram

Table 125. Timing Requirements for Serial Inputs (Passive Mode)

Abbreviated Reference	Parameter	Min	Max	Unit
t80	Clock Period (high to high)	38	—†	ns
t81	Clock Low Time (low to high)	16	—	ns
t82	Clock High Time (high to low)	16	—	ns
t83	Load High Setup (high to high)	8	—	ns
t84	Load Low Setup (low to high)	8	—	ns
t85	Load Hold (high to invalid)	0	—	ns

† Device is fully static; t80 is tested at 200 ns.

Table 126. Timing Characteristics for Serial Outputs (Passive Mode)

Abbreviated Reference	Parameter	Min	Max	Unit
t87	Data Delay (high to valid)	—	20	ns
t88	Enable Data Delay (low to active)	—	20	ns
t89	Disable Data Delay (high to 3-state)	—	20	ns
t90	Data Hold (high to invalid)	1	—	ns
t96	OBE Delay (high to high)	—	20	ns

Timing Characteristics and Requirements (continued)

Simple Serial I/O (continued)
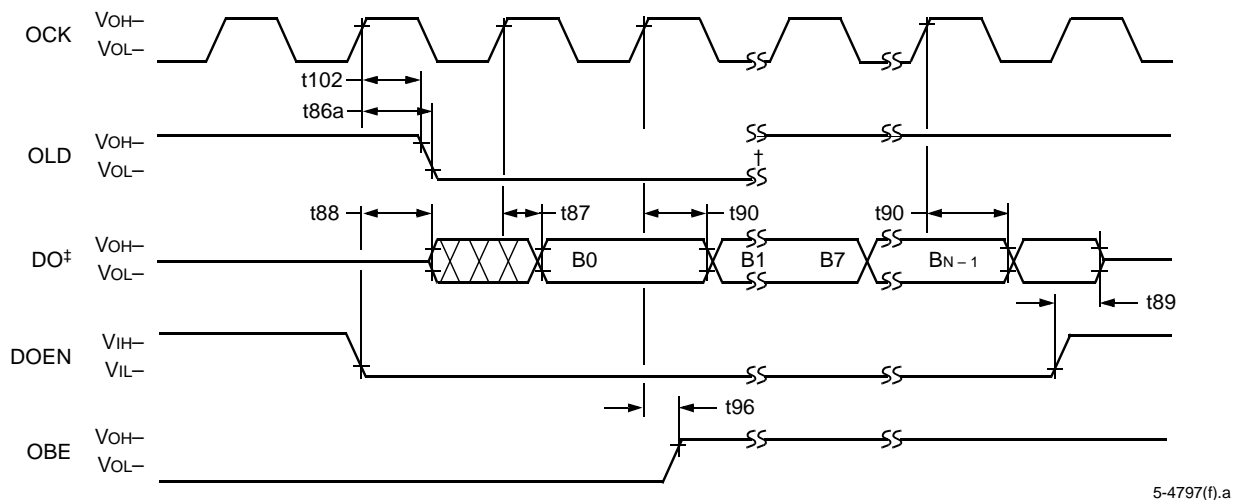
† OLD goes high at the end of bit 6 of 0:15.
‡ See **SSIOC** register, MSB field, to determine if B0 is the MSB or LSB. See **SSIOC** register, ILEN field, to determine if the DO word length is 8 bits or 16 bits. It is assumed that the DODLY bit is 0.

Figure 50. SSIO Active Mode Output Timing Diagram

Table 127. Timing Characteristics for Serial Output (Active Mode)

Abbreviated Reference	Parameter	Min	Max	Unit
t86a	OLD Delay (high to low)	—	20	ns
t102	OLD Hold (high to invalid)	1	—	ns
t87	Data Delay (high to valid)	—	20	ns
t88	Enable Data Delay (low to active)	—	20	ns
t89	Disable Data Delay (high to 3-state)	—	20	ns
t90	Data Hold (high to invalid)	1	—	ns
t96	OBE Delay (high to high)	—	20	ns

Timing Characteristics and Requirements (continued)

Simple Serial I/O (continued)
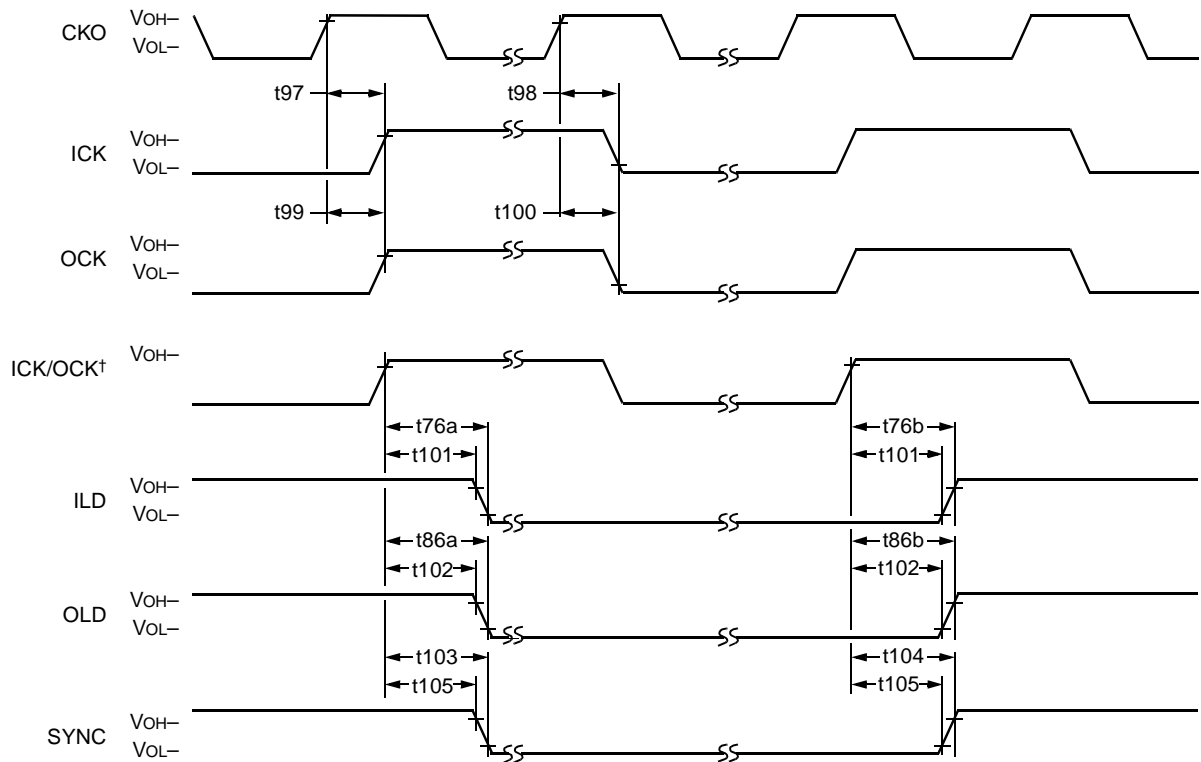
5-4798 (F)

† See **SSIOC** register, LD field.

Figure 51. Serial I/O Active Mode Clock Timing

Table 128. Timing Characteristics for Signal Generation (Active Mode)

Abbreviated Reference	Parameter	Min	Max	Unit
t97	ICK Delay (high to high)	—	15	ns
t98	ICK Delay (high to low)	—	15	ns
t99	OCK Delay (high to high)	—	15	ns
t100	OCK Delay (high to low)	—	15	ns
t76a	ILD Delay (high to low)	—	20	ns
t76b	ILD Delay (high to high)	—	20	ns
t101	ILD Hold (high to invalid)	1	—	ns
t86a	OLD Delay (high to low)	—	20	ns
t86b	OLD Delay (high to high)	—	20	ns
t102	OLD Hold (high to invalid)	1	—	ns
t103	SYNC Delay (high to low)	—	20	ns
t104	SYNC Delay (high to high)	—	20	ns
t105	SYNC Hold (high to invalid)	1	—	ns

Timing Characteristics and Requirements (continued)

Enhanced Serial I/O
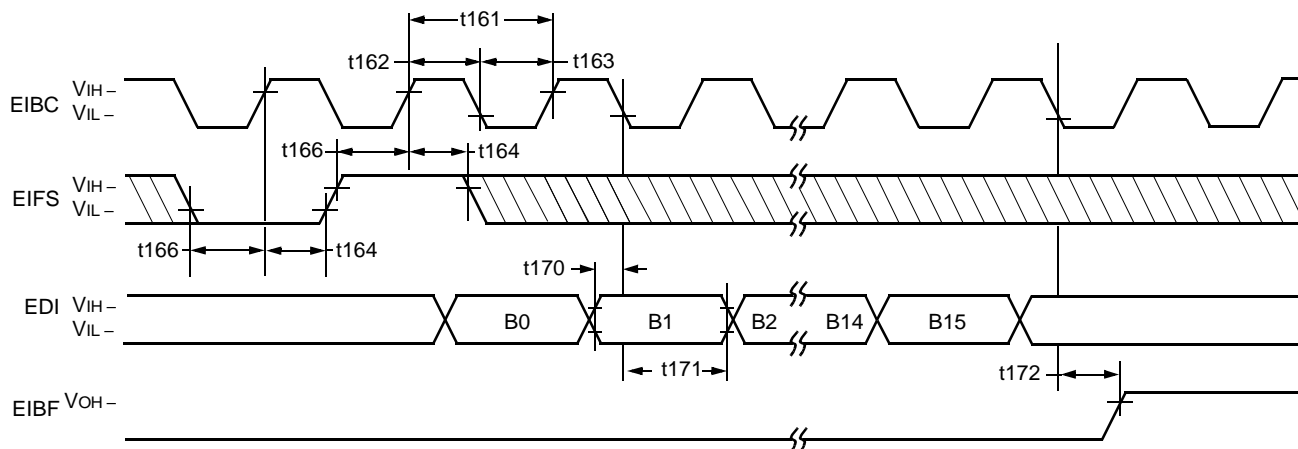
Figure 52. Simple Mode Input Timing Diagram

Note: Simple mode with IMODE = 1, ILEV = 0, ISIZE = 0, ISLEV = 0, and ISDLY = 0.

Table 129. Timing Requirements for ESIO Simple Input Mode

Abbreviated Reference	Parameter	Min	Max	Unit
t161	EIBC Bit Clock Period (high to high)	38	—	ns
t162	EIBC Bit Clock High Time (high to low)	16	—	ns
t163	EIBC Bit Clock Low Time (low to high)	16	—	ns
t164	EIFS Hold Time (high to low or high to high)	8	—	ns
t166	EIFS Setup Time (low to high or high to high)	8	—	ns
t170	EDI Data Setup Time (valid to low)	8	—	ns
t171	EDI Data Hold Time (low to invalid)	4	—	ns

Table 130. Timing Characteristics for ESIO Simple Input Mode

Abbreviated Reference	Parameter	Min	Max	Unit
t172	EIBF Delay (low to high)	—	20	ns

Timing Characteristics and Requirements (continued)Timing Characteristics and Requirements (continued)

Enhanced Serial I/O (continued)
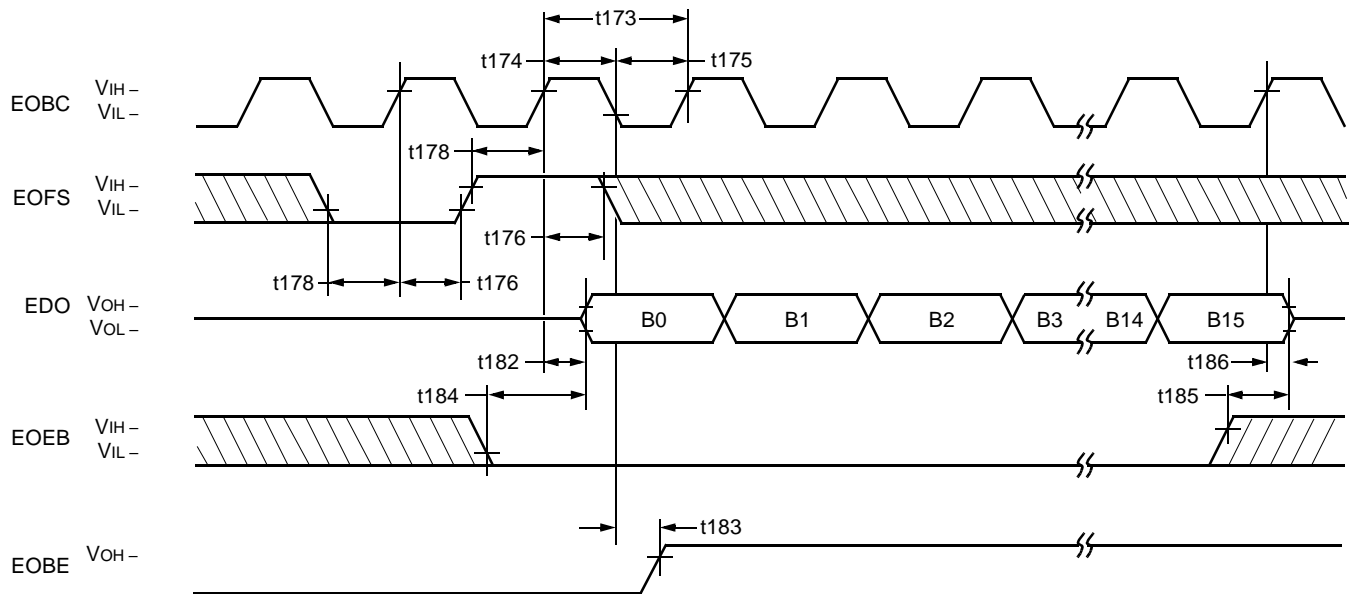
Figure 53. Simple Mode Output Timing Diagram

Note: Simple mode with OMODE = 1, OLEV = 0, OSIZE = 0, and OSLEV = 0.

Table 131. Timing Requirements for ESIO Simple Output Mode

Abbreviated Reference	Parameter	Min	Max	Unit
t173	EOBC Bit Clock Period (high to high)	38	—	ns
t174	EOBC Bit Clock High Time (high to low)	16	—	ns
t175	EOBC Bit Clock Low Time (low to high)	16	—	ns
t176	EOFS Hold Time (high to low or high to high)	8	—	ns
t178	EOFS Setup Time (low to high or high to high)	8	—	ns

Table 132. Timing Characteristics for ESIO Simple Output Mode

Abbreviated Reference	Parameter	Min	Max	Unit
t182[†]	EDO Data Delay (high to valid)	—	20	ns
t183	EOBE Delay (low to high)	—	20	ns
t184[†]	External Enable Data (low to active)	—	20	ns
t185[‡]	External Disable Data (high to 3-state)	—	20	ns
t186[‡]	EDO 3-state Delay (high to 3-state)	—	20	ns

† EDO is active after the t182 interval or t184 interval, whichever occurs last.
‡ EDO is inactive after the t185 interval or t186 interval, whichever occurs first.

Timing Characteristics and Requirements (continued)

Enhanced Serial I/O (continued)
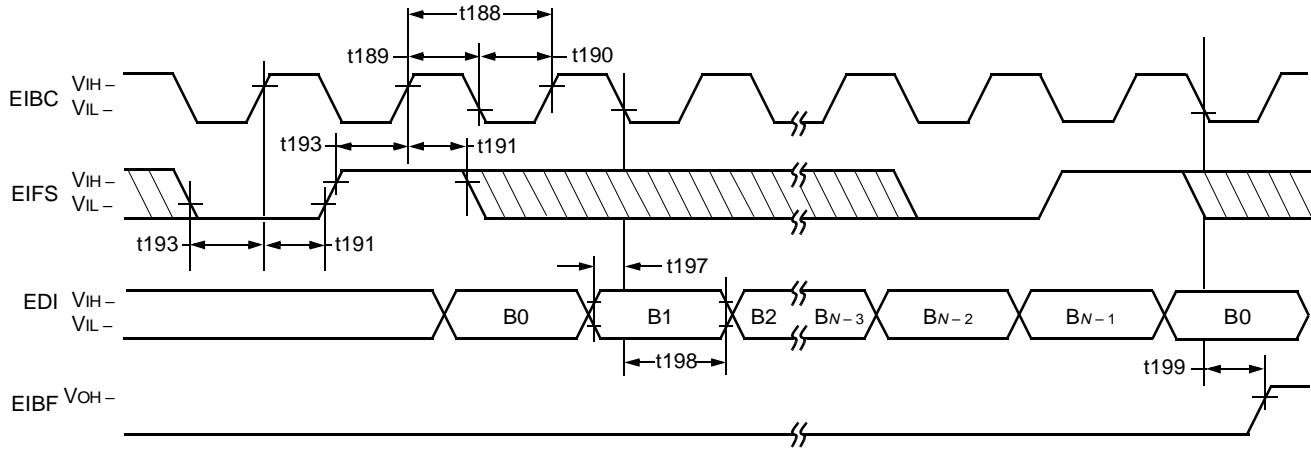
Figure 54. Frame Mode Input Timing Diagram

Note: Frame mode with IMODE = 0, ILEV = 0, ISIZE = 0, ISLEV = 0, and ISDLY = 0, and IFRMSZ[1:0] = N.

Table 133. Timing Requirements for ESIO Frame Input Mode

Abbreviated Reference	Parameter	Min	Max	Unit
t188	EIBC Bit Clock Period (high to high)	122	—	ns
t189	EIBC Bit Clock High Time (high to low)	16	—	ns
t190	EIBC Bit Clock Low Time (low to high)	16	—	ns
t191	EIFS Hold Time (high to low or high to high)	8	—	ns
t193	EIFS Setup Time (low to high or high to high)	8	—	ns
t197	EDI Data Setup Time (valid to low)	8	—	ns
t198	EDI Data Hold Time (low to invalid)	4	—	ns

Table 134. Timing Characteristics for ESIO Frame Input Mode

Abbreviated Reference	Parameter	Min	Max	Unit
t199	EIBF Delay (low to high)	—	20	ns

Timing Characteristics and Requirements (continued)

Enhanced Serial I/O (continued)
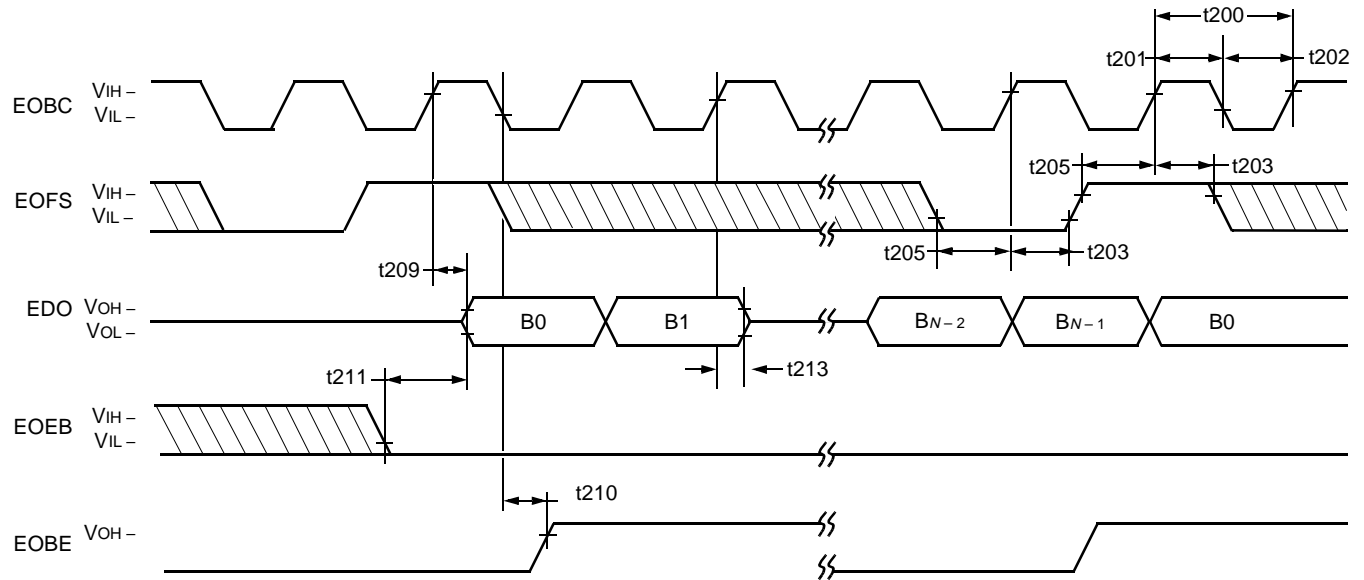
Figure 55. Frame Mode Output Timing Diagram

Note: Frame mode with OMODE = 0, OLEV = 0, OSIZE = 0, OSLEV = 0, and OFRMSZ[1:0] = N.

Table 135. Timing Requirements for ESIO Frame Output Mode

Abbreviated Reference	Parameter	Min	Max	Unit
t200	EOBC Bit Clock Period (high to high)	122	—	ns
t201	EOBC Bit Clock High Time (high to low)	16	—	ns
t202	EOBC Bit Clock Low Time (low to high)	16	—	ns
t203	EOFS Hold Time (high to low or high to high)	8	—	ns
t205	EOFS Setup Time (low to high or high to high)	8	—	ns

Table 136. Timing Characteristics for ESIO Frame Output Mode

Abbreviated Reference	Parameter	Min	Max	Unit
t209[†]	EDO Data Delay (high to valid)	—	20	ns
t210	EOBE Delay (low to high)	—	20	ns
t211[†]	EOEB External Enable Delay (low to active)	—	20	ns
t213	EDO 3-state Delay (high to 3-state)	—	20	ns

† EDO is active after the t209 interval or t211 interval, whichever occurs last.

Outline Diagrams

144-Pin TQFP Outline Diagram

All dimensions are in millimeters.

5-3815 (F)

Notes

July 2000
DS98-032WTEC

microelectronics group

Lucent Technologies
Bell Labs Innovations

Lucent Technologies
Bell Labs Innovations

Significant Changes to the DSP16210 Digital Signal Processor Data Sheet Since January 1999

Page(s)	Change
20	The last paragraph on this page had stated that writing a 1 to a bit in the **ins** register resets the corresponding interrupt source. This statement was not correct and has been modified.
24	The first paragraph on this page had stated that a separate vector, TRAP, is provided for the pin trap. The name of this vector is actually PTRAP.
44	In Table 18, MIOU⟨0,1⟩ Command (mcmd⟨0,1⟩) Register, the INPT_DS command was missing. It has been added.
48, 86	In Table 21, MIOU Command Latencies, the register read-write latencies for the **morp⟨0,1⟩** and **miwp⟨0,1⟩** registers were removed. This is because these latencies are now automatically compensated by the DSP16000 assembler. This is explained in Peripheral Register Write-Read Latency.
78	In Table 34, Overall Replacement Table, definitions for the new symbols aDEE, aSEE, and aTEE were added. This agrees with the symbols in Table 32, Instruction Set Summary, Table 36, F1E Function Statement Syntax, and with the *DSP16000 Digital Signal Processor Core* Information Manual.
91	The description of the **alf** register was expanded.
127 130 131	In Table 81, Command Encoding for Boot Routines, the following changes were made: The command code 0x98 was added. Corrections were made to entries in the PODS/PDS column for command codes 0x12, 0x52, 0x13, 0x53, 0x14, and 0x54. A correction was made to the description for command code 0x15. Also, the following command codes were added: 0x55, 0x95, and 0xD5.