Advisory
May 1999

microelectronics group

Lucent Technologies
Bell Labs Innovations

# Clarification to the Serial I/O Control Register Description for the DSP1620/27/28/29 Devices

## Active Clock Frequency

The purpose of this advisory is to clarify the function of the serial I/O control registers in the DSP1620/27/28/29 devices. Specifically, it clarifies the function of the control register field that specifies the active clock frequency. The device data sheets state that the active clock frequency is a ratio of the **input** clock frequency on the CKI pin (DSP1627/28/29 devices) or the output clock frequency on the CKO pin (DSP1620 device). For all four devices, the actual active clock frequency is a ratio of the **internal** clock frequency, which can be programmed as either the input clock frequency on the CKI pin or the output of an internal clock synthesizer (PLL).

Table 1 summarizes information for each of the four devices. It lists the document number for each device data sheet. For example, the data sheet for the DSP1620, entitled *DSP1620 Digital Signal Processor*, has the document number DS97-321WDSP. Table 1 also lists the name of each serial I/O unit on each device, the corresponding control register, the data sheet page number that describes the register, and the corresponding field within the register that specifies the active clock frequency. For example, the DSP1620 contains two serial I/O units named SIO and SSIO. The control register for SIO is **sioc** described on page 94 of the data sheet. Bits 8—7 within **sioc** (CLK1 field) specify the active clock frequency of the SIO.

**Table 1. Data Sheet and Serial I/O Information for the DSP1620/27/28/29 Devices**

| Device | Data Sheet Document Number | Serial I/O Units | | | | |
|---|---|---|---|---|---|---|
| | | Name | Control Register | Data Sheet Page No. | Active Clock Frequency Control Field | |
| | | | | | Bits | Name |
| DSP1620 | DS97-321WDSP | SIO | **sioc** | 94 | 8—7 | CLK1 |
| | | SSIO | **SSIOC** | 96 | 8—7 | CLK2 |
| DSP1627 | DS96-188WDSP | SIO | **sioc** | 45 | 8—7 | CLK |
| | | SIO2 | | | | |
| DSP1628 | DS97-040WDSP | SIO | **sioc** | 55 | 8—7 | CLK |
| | | SIO2 | | | | |
| DSP1629 | DS96-039WDSP | SIO | **sioc** | 46 | 8—7 | CLK |
| | | SIO2 | | | | |

Table 2 shows a corrected description of the CLK/CLK1/CLK2 field of the serial I/O control register. The specific correction is shown in bold type—the active clock frequency is a ratio of $f_{\text{internal clock}}$, not of CKI or CKO.

**Table 2. Corrected Description of CLK/CLK1/CLK2 Field**

| Field | Value | Description |
|---|---|---|
| CLK<br>CLK1<br>CLK2 | 00 | Active clock frequency = $\mathbf{f_{internal\ clock}} \div 2$ |
| | 01 | Active clock frequency = $\mathbf{f_{internal\ clock}} \div 6$ |
| | 10 | Active clock frequency = $\mathbf{f_{internal\ clock}} \div 8$ |
| | 11 | Active clock frequency = $\mathbf{f_{internal\ clock}} \div 10$ |

For additional information, contact your Microelectronics Group Account Manager or the following:

| | |
|---|---|
| INTERNET: | **http://www.lucent.com/micro** |
| E-MAIL: | **docmaster@micro.lucent.com** |
| N. AMERICA: | Microelectronics Group, Lucent Technologies Inc., 555 Union Boulevard, Room 30L-15P-BA, Allentown, PA 18103 **1-800-372-2447**, FAX 610-712-4106 (In CANADA: **1-800-553-2448**, FAX 610-712-4106) |
| ASIA PACIFIC: | Microelectronics Group, Lucent Technologies Singapore Pte. Ltd., 77 Science Park Drive, #03-18 Cintech III, Singapore 118256 **Tel. (65) 778 8833**, FAX (65) 777 7495 |
| CHINA: | Microelectronics Group, Lucent Technologies (China) Co., Ltd., A-F2, 23/F, Zao Fong Universe Building, 1800 Zhong Shan Xi Road, Shanghai 200233 P. R. China **Tel. (86) 21 6440 0468**, **ext. 316**, FAX (86) 21 6440 0652 |
| JAPAN: | Microelectronics Group, Lucent Technologies Japan Ltd., 7-18, Higashi-Gotanda 2-chome, Shinagawa-ku, Tokyo 141, Japan **Tel. (81) 3 5421 1600**, FAX (81) 3 5421 1700 |
| EUROPE: | Data Requests: MICROELECTRONICS GROUP DATALINE: **Tel. (44) 1189 324 299**, FAX (44) 1189 328 148 Technical Inquiries: GERMANY: **(49) 89 95086 0** (Munich), UNITED KINGDOM: **(44) 1344 865 900** (Ascot), FRANCE: **(33) 1 40 83 68 00** (Paris), SWEDEN: **(46) 8 594 607 00** (Stockholm), FINLAND: **(358) 9 4354 2800** (Helsinki), ITALY: **(39) 02 6608131** (Milan), SPAIN: **(34) 1 807 1441** (Madrid) |

microelectronics group

May 1999
AY99-001WDSP
(must accompany DS97-321WDSP, DS96-188WDSP, DS97-040WDSP, and DS96-039WDSP)

Lucent Technologies
Bell Labs Innovations

Data Sheet
June 1998

microelectronics group

Lucent Technologies
Bell Labs Innovations

# DSP1620 Digital Signal Processor

## 1 Features

- Optimized for digital cellular infrastructure applications—equalization, channel coding, speech coding
- Large, on-chip DPRAM (32 Kwords) that eliminates need for fast external SRAM
- ECCP for efficient equalization and channel coding
- DMA-based I/O that minimizes DSP core overhead for I/O processing
- Bit manipulation unit for higher coding efficiency
- On-chip programmable, PLL (phase-lock loop) clock synthesizer enables low-cost, low-power implementations
- Instruction cycle times:
  —At 3 V are 8.3 ns (120 MIPS), 10.0 ns (100 MIPS), and 11.1 ns (90 MIPS)
  —At 5 V is 11.1 ns (90 MIPS)
- Low system power consumption with flexible power management modes
- Support for 128 Kwords external memory access
- Four external vectored interrupts
- 25 Mbits/s serial I/O port (SIO) with multiprocessor capability—16-bit data channel, 8-bit protocol channel
- Two dedicated DMA controllers (MIOUs) to off-load I/O processing from DSP core
- 25 Mbits/s simple serial I/O port (SSIO) coupled with MIOU0 to support low-overhead DMA-based I/O
- 16-bit parallel host interface (PHIF16) coupled with MIOU1 to support low-overhead DMA-based I/O
  —Supports 8- or 16-bit external bus configurations
  —Supports 8- or 16-bit logical transfers in 8-bit external configuration
  —*Motorola** or *Intel*† compatible
- Memory sequencer for single-instruction access to both X and Y external memory space
- Single-cycle squaring feature increases coding efficiency
- 16 x 16-bit multiplication and 36-bit accumulation in one instruction cycle for efficient algorithm implementations
- Instruction cache for high-speed, program-efficient, zero-overhead looping
- 8-bit control I/O interface provides increased flexibility and lower system costs
- 256 memory-mapped I/O ports for interfacing flexibility
- *IEEE*‡ P1149.1 test port (JTAG with boundary scan)
- Full-speed in-circuit emulation hardware development system on-chip for faster system developments
- Supported by DSP1620 software and hardware development tools
- On-chip boot routines for flexible downloading
- 132-pin BQFP package and 144-pin TQFP package

## 2 Description

The DSP1620 is a DSP1600 core-based fixed-point digital signal processor with a large amount of on-chip RAM and a flexible DMA-based I/O structure that is designed specifically for digital cellular infrastructure applications. This device also contains a bit manipulation unit (BMU) and an error correction coprocessor (ECCP) for enhanced signal coding efficiency. The DSP1620 offers 120, 100, or 90 MIPS performance at 3 V and 90 MIPS performance at 5 V.

The large, 32 Kword on-chip, dual-port RAM (DPRAM) supports downloadable system design—a must for wireless infrastructure—to support field upgrades for evolving digital cellular standards. The DSP1620 can address 30 Kwords of on-chip DPRAM and up to 64 Kwords of external storage in its code and coefficient memory address space. In addition, the DSP1620 can address 32 Kwords of on-chip DPRAM and up to 128 Kwords of external storage in its external memory address space (64 Kwords/Data and 64 Kwords/Program).

To optimize I/O throughput and reduce the I/O service routine burden on the DSP core, the DSP1620 is equipped with two modular I/O units (MIOUs) that manage one of the serial ports (SSIO) and the 16-bit parallel host interface (PHIF16) peripherals. The MIOUs provide transparent DMA transfers between the peripherals and on-chip DPRAM.

The error correction coprocessor is a powerful hardware engine for Viterbi decoding with instructions for maximum likelihood sequence estimation (MLSE) equalization and convolutional decoding.

The combination of a large, on-chip RAM, 120 MIPS performance, and efficient I/O management makes the DSP1620 an ideal solution for supporting multiple channels of voice and data traffic in digital cellular infrastructure equipment.

The device is packaged in a 132-pin BQFP and a 144-pin TQFP; it is available with 11.1 ns instruction cycle speed at 5 V and 8.3 ns, 10.0 ns, and 11.1 ns instruction cycle speeds at 3 V.

* *Motorola* is a registered trademark of Motorola, Inc.
† *Intel* is a registered trademark of Intel Corp.
‡ *IEEE* is a registered trademark of The Institute of Electrical and Electronics Engineers, Inc.

# Table of Contents

# List of Figures

**Figures** **Page**

## List of Figures (continued)

# List of Tables

## List of Tables (continued)

## List of Tables (continued)

**Tables** **Page**

## List of Tables (continued)

**Tables** **Page**

## 3 Pin Information

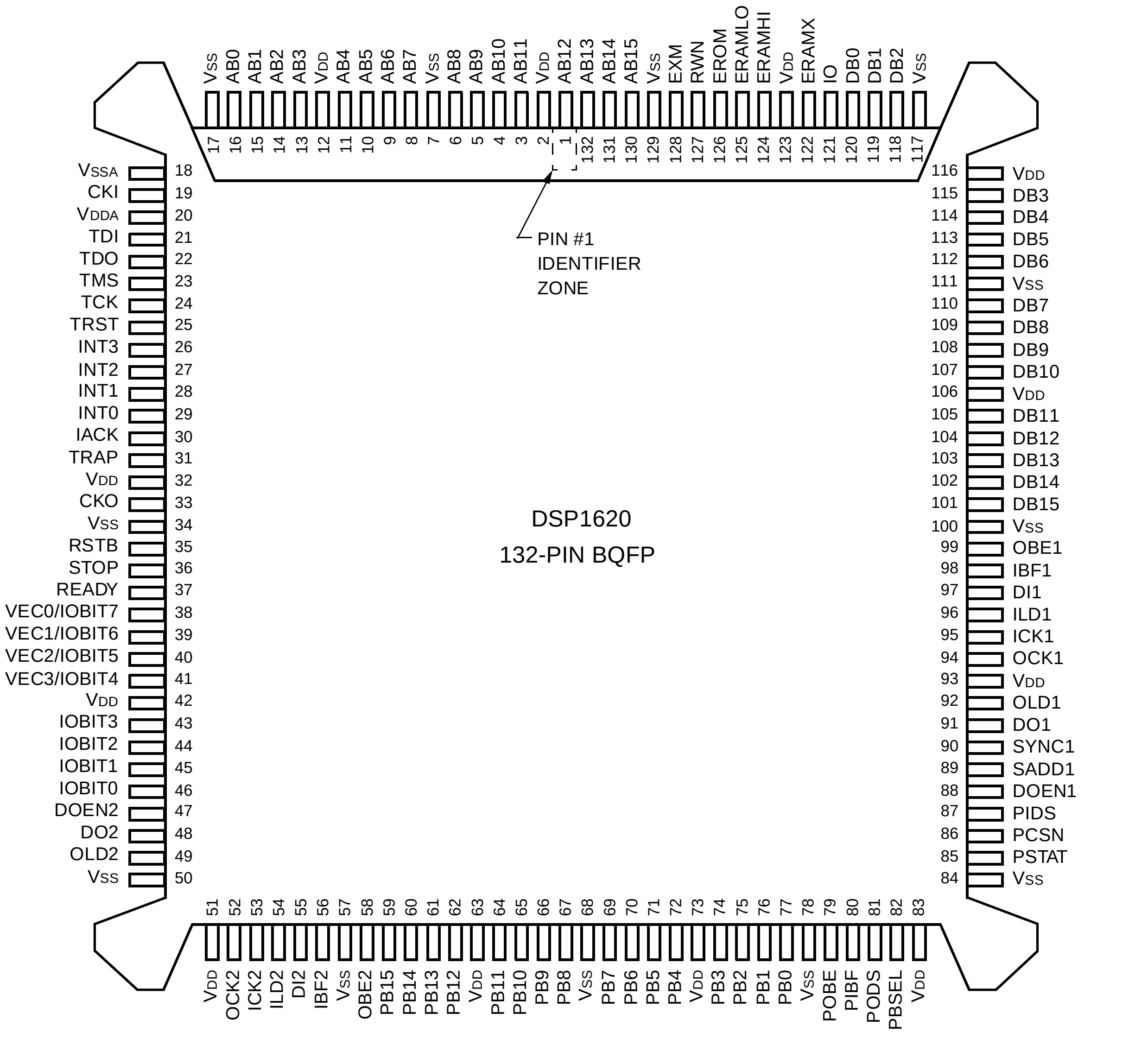
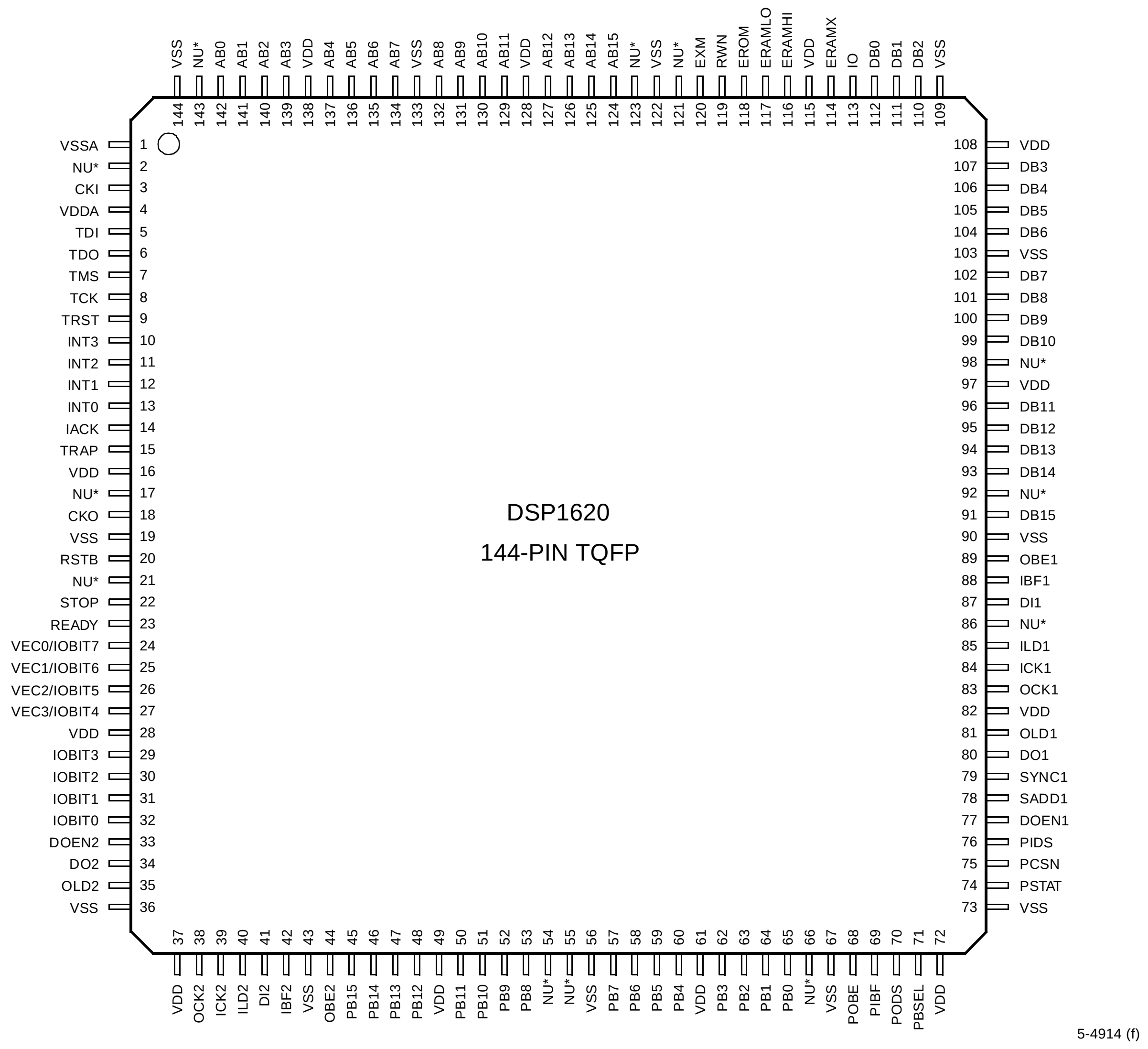

DSP1620
132-PIN BQFP
PIN #1
IDENTIFIER
ZONE
1 VDD 2 AB11 3 AB10 4 AB9 5 AB8 6 VSS 7 AB7 8 AB6 9 AB5 10 AB4 11 VDD 12 AB3 13 AB2 14 AB1 15 AB0 16 VSS 17
18 VSSA 19 CKI 20 VDDA 21 TDI 22 TDO 23 TMS 24 TCK 25 TRST 26 INT3 27 INT2 28 INT1 29 INT0 30 IACK 31 TRAP 32 VDD 33 CKO 34 VSS 35 RSTB 36 STOP 37 READY 38 VEC0/IOBIT7 39 VEC1/IOBIT6 40 VEC2/IOBIT5 41 VEC3/IOBIT4 42 VDD 43 IOBIT3 44 IOBIT2 45 IOBIT1 46 IOBIT0 47 DOEN2 48 DO2 49 OLD2 50 VSS
51 VDD 52 OCK2 53 ICK2 54 ILD2 55 DI2 56 IBF2 57 VSS 58 OBE2 59 PB15 60 PB14 61 PB13 62 PB12 63 VDD 64 PB11 65 PB10 66 PB9 67 PB8 68 VSS 69 PB7 70 PB6 71 PB5 72 PB4 73 VDD 74 PB3 75 PB2 76 PB1 77 PB0 78 VSS 79 POBE 80 PIBF 81 PODS 82 PBSEL 83 VDD
84 VSS 85 PSTAT 86 PCSN 87 PIDS 88 DOEN1 89 SADD1 90 SYNC1 91 DO1 92 OLD1 93 VDD 94 OCK1 95 ICK1 96 ILD1 97 DI1 98 IBF1 99 OBE1 100 VSS 101 DB15 102 DB14 103 DB13 104 DB12 105 DB11 106 VDD 107 DB10 108 DB9 109 DB8 110 DB7 111 VSS 112 DB6 113 DB5 114 DB4 115 DB3 116 VDD
117 VSS 118 DB2 119 DB1 120 DB0 121 IO 122 ERAMX 123 VDD 124 ERAMHI 125 ERAMLO 126 EROM 127 RWN 128 EXM 129 VSS 130 AB15 131 AB14 132 AB13 AB12
5-4773(F)

**Figure 1. DSP1620 132-Pin BQFP Pin Diagram (Top View)**

## 3 Pin Information (continued)



* NU = Not Usable; no external connections are allowed.

**Figure 2. DSP1620 144-Pin TQFP Pin Diagram (Top View)**

## 3 Pin Information (continued)

Functional descriptions of BQFP pins 1—132 and TQFP pins 1—144 are found in Section 6, Signal Descriptions. 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, essentially no dc current is drawn. Although input and I/O buffers may be left untied, the guidelines for terminating unused pins are as follows:

- NC (no connect) pins should be left floating.
- Input pins can either be tied directly to VSS or tied to VDD through a 10 kΩ resistor. Deciding VSS or VDD is important for input pins with special functions. For example, if the PHIF16 port is unused then the PCSN (PHIF16 Chip Select Not) pin should be tied high (no select).
- Output pins should be left floating.
- Bidirectional I/O pins configured as inputs should be tied to VDD or VSS through a 10 kΩ resistor. Bidirectional I/O pins configured as outputs should be left floating. Bit I/O pins are programmed as inputs when the device is reset (**sbit** = 0x00xx). **Do not directly connect them to VSS or VDD because of the output buffer.**

**Table 1. Pin Descriptions**

| BQFP Pin | TQFP Pin | Symbol | Type | Name/Function |
|---|---|---|---|---|
| 101, 102, 103, 104, 105, 107, 108, 109, 110, 112, 113, 114, 115, 118, 119, 120 | 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. |
| 121 | 113 | IO | O† | Data Address 0x8000 to 0x80FF I/O Enable. |
| 124 | 116 | ERAMHI | O† | Data Address 0xC000 to 0xFFFF External RAM Enable. |
| 125 | 117 | ERAMLO | O† | Data Address 0x8100 to 0xBFFF External RAM Enable. |
| 122 | 114 | ERAMX | O† | Data Address 0x8000 to 0xFFFF External RAM Enable. |
| 126 | 118 | EROM | O† | Program Address External ROM Enable. |
| 127 | 119 | RWN | O† | Read/Write Not. |
| 128 | 120 | EXM | I | External ROM Enable. |
| 130, 131, 132, 1, 3, 4, 5, 6, 8, 9, 10, 11, 13, 14, 15, 16 | 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. |
| 26, 27, 28, 29 | 10, 11, 12, 13 | INT[3:0] | I | Vectored Interrupts INT3, INT2, INT1, and INT0. |
| 30 | 14 | IACK | O* | Interrupt Acknowledge. |
| 36 | 22 | STOP | I | STOP Input Clock (negative assertion). |
| 37 | 23 | READY | I | Processing Enable. |
| 31 | 15 | TRAP | I/O* | Nonmaskable Program Trap/Breakpoint Indication. |
| 35 | 20 | RSTB | I | Reset (negative assertion). |
| 33 | 18 | CKO | O† | Processor Clock Output. |
| 24 | 8 | TCK | I | JTAG Test Clock. |

* 3-states when RSTB = 0 or by JTAG control.
† 3-states when the level of RSTB = 0 and INT0 = 1. Output = 1 when the level of RSTB = 0 and INT0 = 0, except CKO which is free-running.
‡ 3-states by JTAG control.
§ Pull-up devices on input.
** 3-states when RSTB = 0, JTAG control, or **PHIFC** register bit PCFIG = 0.
†† For SIO multiprocessor applications, add 5 kΩ external pull-up resistors to SADD1 for proper initialization.

## 3 Pin Information (continued)

**Table 1. Pin Descriptions** (continued)

| BQFP Pin | TQFP Pin | Symbol | Type | Name/Function |
|---|---|---|---|---|
| 23 | 7 | TMS | I§ | JTAG Test Mode Select. |
| 22 | 6 | TDO | O‡ | JTAG Test Data Output. |
| 21 | 5 | TDI | I§ | JTAG Test Data Input. |
| 25 | 9 | TRST | I§ | JTAG Test Reset (negative assertion). |
| 19 | 3 | CKI | I | Clock Input. |
| 38 | 24 | VEC0/IOBIT7 | I/O* | Vectored Interrupt Bit 0/BIO Signal Bit 7. |
| 39 | 25 | VEC1/IOBIT6 | I/O* | Vectored Interrupt Bit 1/BIO Signal Bit 6. |
| 40 | 26 | VEC2/IOBIT5 | I/O* | Vectored Interrupt Bit 2/BIO Signal Bit 5. |
| 41 | 27 | VEC3/IOBIT4 | I/O* | Vectored Interrupt Bit 3/BIO Signal Bit 4. |
| 43 | 29 | IOBIT3 | I/O* | BIO Signal Bit 3. |
| 44 | 30 | IOBIT2 | I/O* | BIO Signal Bit 2. |
| 45 | 31 | IOBIT1 | I/O* | BIO Signal Bit 1. |
| 46 | 32 | IOBIT0 | I/O* | BIO Signal Bit 0. |
| 47 | 33 | DOEN2 | I | SSIO Data Output Enable. |
| 55 | 41 | DI2 | I | SSIO Data Input. |
| 53 | 39 | ICK2 | I/O* | SSIO Input Clock. |
| 58 | 44 | OBE2 | O* | SSIO Output Buffer Empty. |
| 56 | 42 | IBF2 | O* | SSIO Input Buffer Full. |
| 49 | 35 | OLD2 | I/O* | SSIO Output Load. |
| 54 | 40 | ILD2 | I/O* | SSIO Input Load. |
| 48 | 34 | DO2 | O* | SSIO Data Output. |
| 52 | 38 | OCK2 | I/O* | SSIO Output Clock. |
| 88 | 77 | DOEN1 | I/O* | SIO Data Output Enable. |
| 89 | 78 | SADD1†† | I/O* | SIO Multiprocessor Address. |
| 90 | 79 | SYNC1 | I/O* | SIO Multiprocessor Synchronization. |
| 91 | 80 | DO1 | O* | SIO Data Output. |
| 92 | 81 | OLD1 | I/O* | SIO Output Load. |
| 94 | 83 | OCK1 | I/O* | SIO Output Clock. |
| 95 | 84 | ICK1 | I/O* | SIO Input Clock. |
| 96 | 85 | ILD1 | I/O* | SIO Input Load. |
| 97 | 87 | DI1 | I | SIO Data Input. |
| 98 | 88 | IBF1 | O* | SIO Input Buffer Full. |
| 99 | 89 | OBE1 | O* | SIO Output Buffer Empty. |
| 59, 60, 61, 62, 64, 65, 66, 67 | 45, 46, 47, 48, 50, 51, 52, 53 | PB[15:8] | I/O** | PHIF16 Data Bus 15—8. |
| 69, 70, 71, 72, 74, 75, 76, 77 | 57, 58, 59, 60, 62, 63, 64, 65 | PB[7:0] | I/O* | PHIF16 Data Bus 7—0. |
| 79 | 68 | POBE | O* | PHIF16 Output Buffer Empty. |
| 80 | 69 | PIBF | O* | PHIF16 Input Buffer Full. |
| 81 | 70 | PODS | I | PHIF16 Output Data Strobe. |

* 3-states when RSTB = 0, or by JTAG control.
† 3-states when the level of RSTB = 0 and INT0 = 1. Output = 1 when the level of RSTB = 0 and INT0 = 0, except CKO which is free-running.
‡ 3-states by JTAG control.
§ Pull-up devices on input.
** 3-states when RSTB = 0, JTAG control, or **PHIFC** register bit PCFIG = 0.
†† For SIO multiprocessor applications, add 5 kΩ external pull-up resistors to SADD1 for proper initialization.

## 3 Pin Information (continued)

**Table 1. Pin Descriptions** (continued)

| BQFP Pin | TQFP Pin | Symbol | Type | Name/Function |
|---|---|---|---|---|
| 87 | 76 | PIDS | I | PHIF16 Input Data Strobe. |
| 82 | 71 | PBSEL | I | PHIF16 (8-bit external mode) Byte Select. |
| 85 | 74 | PSTAT | I | PHIF16 Status Register Select. |
| 86 | 75 | PCSN | I | PHIF16 Chip Select Not. |
| 7, 17, 34, 50, 57, 68, 78, 84, 100, 111, 117, 129 | 19, 36, 43, 56, 67, 73, 90, 103, 109, 122, 133, 144 | $V_{SS}$ | P | Ground. |
| 2, 12, 32, 42, 51, 63, 73, 83, 93, 106, 116, 123 | 16, 28, 37, 49, 61, 72, 82, 97, 108, 115, 128, 138 | $V_{DD}$ | P | Power Supply. |
| 20 | 4 | $V_{DDA}$ | P | Analog Power Supply. |
| 18 | 1 | $V_{SSA}$ | P | Analog Ground. |
| — | 2, 17, 21, 54, 55, 66, 86, 92, 98, 121, 123, 143 | NU | — | Not Usable (No external connections are allowed.) |

* 3-states when RSTB = 0, or by JTAG control.
† 3-states when the level of RSTB = 0 and INT0 = 1. Output = 1 when the level of RSTB = 0 and INT0 = 0, except CKO which is free-running.
‡ 3-states by JTAG control.
§ Pull-up devices on input.
** 3-states when RSTB = 0, JTAG control, or **PHIFC** register bit PCFIG = 0.
†† For SIO multiprocessor applications, add 5 kΩ external pull-up resistors to SADD1 for proper initialization.

# 4 Hardware Architecture

The DSP1620 device is a 16-bit, fixed-point, programmable digital signal processor (DSP). The DSP1620 consists of an enhanced DSP1600 core together with on-chip memory and peripherals. Added architectural features give the DSP1620 high program efficiency for signal coding and I/O-intensive applications.

Throughout this manual, all DSP registers directly writable or readable by DSP instructions are printed in lower-case. I/O pins and nonprogram-accessible registers are upper-case. All register names and DSP instructions are printed in **boldface** when written in text descriptions.

## 4.1 DSP1620 Architectural Overview

Figure 3 shows a block diagram of the DSP1620. The following blocks make up this device.

### DSP1600 Core

The DSP1600 core is the heart of the DSP1620 chip. The core contains data and address arithmetic units, and control for on-chip memory and peripherals. The core provides support for external memory wait-states and on-chip dual-port RAM and features vectored interrupts and a trap mechanism. The core is discussed further in Section 4.2.

### Dual-Port RAM (DPRAM)

This block contains 30 banks (banks 1—30) of zero wait-state memory. Each bank consists of 1K 16-bit words and has separate address and data ports to the instruction/coefficient and data memory spaces. A program can reference memory from either space. The DSP1600 core automatically performs the required multiplexing. If references to both ports of a single bank are made simultaneously, the DSP1600 core automatically inserts a wait-state and performs the data port access first, followed by the instruction/coefficient port access.

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. DPRAM can also be used for downloading self-test code via the JTAG port.

When the ECCP is active, DPRAM bank 30 is dedicated to the ECCP (for storing traceback information) and cannot be accessed by the core.

### IORAM

IORAM storage consists of two 1 Kword banks (banks 31 and 32) of on-chip DPRAM that resides in the core's internal data memory space. Each bank of IORAM has two data and two 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. If both the core and MIOU simultaneously access the same IORAM bank, the DSP1600 core automatically inserts a wait-state and performs the MIOU access first, followed by the core access. MIOU IORAM requests that do not collide with core IORAM requests do not incur a wait-state.

MIOU0 (controls SSIO) is attached to RAM bank 32; MIOU1 (controls PHIF16) is attached to RAM bank 31. Portions of IORAM not dedicated to I/O processing can be used as general-purpose DPRAM in the data memory map.

### Read-Only Memory (ROM)

The DSP1620 contains a 4 Kword boot ROM. The boot routines are detailed in Section 7.

### External Memory Interface (EMI)

The EMI is used to connect the DSP1620 to external memory and I/O devices. It supports read/write operations from/to instruction/coefficient memory (X memory space) and data memory (Y memory space). The DSP1600 core automatically controls the EMI. Instructions can transparently reference external memory from either set of internal buses. A sequencer allows a single instruction to access both the X and the Y external memory spaces.

### Clock Synthesis

The DSP powers up with a 1X input clock (CKI) as the source for the processor clock. An on-chip clock synthesizer (PLL) can also be used to generate the system clock for the DSP that runs at a frequency multiple of the input clock. The clock synthesizer is deselected and powered down on reset. For low-power operation, an internally generated slow clock can drive the DSP. If both the clock synthesizer and the internally generated slow clock are selected, the slow clock drives the DSP; however, the synthesizer continues to run.

The clock synthesizer and other programmable clock sources are discussed in Section 4.16. The use of these programmable clock sources for power management is discussed in Section 4.17.

## 4 Hardware Architecture (continued)

### Bit Manipulation Unit (BMU)

The BMU extends the DSP1600 core instruction set to provide more efficient bit operations on accumulators. The BMU contains logic for barrel shifting, normalization, and bit-field insertion/extraction. The unit also contains a set of 36-bit alternate accumulators. The data in the alternate accumulators can be shuffled with the data in the main accumulators. Flags returned by the BMU are testable by the DSP1600 conditional instructions.

### Bit I/O Unit (BIO)

The BIO provides convenient and efficient monitoring and control of eight individually configurable pins. 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 mesh seamlessly with conditional instructions.

### Serial I/O Unit (SIO)

The SIO offers an asynchronous, full-duplex, double-buffered channel that operates at up to 25 Mbits/s (in a nonmultiprocessor configuration), and easily interfaces with other Lucent Technologies fixed-point DSPs in a multiple-processor environment (multiprocessor mode). Commercially available codecs and time-division multiplex (TDM) channels can be interfaced to the SIO with few, if any, additional components.

In multiprocessor mode, an 8-bit serial protocol channel can be transmitted in addition to the address of the called processor. This feature is useful for transmitting high-level framing information or for error detection and correction.

### Simple Serial I/O Unit (SSIO)

The SSIO offers an asynchronous, full-duplex, double-buffered external channel that operates up to 25 Mbits/s. Commercially available codecs and time-division multiplex channels can be interfaced to the SSIO with few, if any, additional components. The SSIO external interface is identical to the SIO external interface with the multiprocessor mode functionality and SADD and SYNC signals deleted.

The SSIO is a DMA peripheral that interfaces directly to the core's data memory space under the control of MIOU0.

## 4 Hardware Architecture (continued)

### Parallel Host Interface (PHIF16)

The PHIF16 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 fixed point DSPs (e.g., DSP1611, DSP1616, DSP1617, DSP1618, DSP1620, DSP1627, DSP1628, DSP1629), microprocessors, or peripheral 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 Section 4.10, Parallel Host Interface (PHIF16)).

PHIF16 is a DMA peripheral and interfaces directly to the core's data memory space under the control of MIOU1.

### Timer

The timer 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 timer can be stopped and restarted at any time.

### JTAG and HDS Module

The on-chip Hardware Development System (HDS) performs instruction breakpointing and branch tracing at full speed without additional off-chip hardware. Using the JTAG port, breakpointing is set up, and the trace history is read back. The port works in conjunction with the HDS code in the on-chip ROM and the hardware and software in a remote computer.

A maximum of four hardware breakpoints can be set on instruction addresses. A counter can be preset with the number of breakpoints to receive before trapping the core. Breakpoints can be set in interrupt service routines. Alternately, the counter can be preset with the number of cache instructions to execute before trapping the core.

Every time the program branches (rather than executing the next sequential instruction) the addresses of the instructions executed before and after the branch are captured in circular memory. This memory contains the last four pairs of program discontinuities for hardware tracing.

In systems with multiple processors, the DSPs can be configured so that any processor reaching a breakpoint causes all the other processors to be trapped (see Section 4.3, Interrupts and Trap).

### Pin Multiplexing

Upon reset, the vectored interrupt indication signals, VEC[3:0], are connected to the package pins while IOBIT[4:7] are disconnected. Setting bit 12, EBIOH, of the **ioc** register connects IOBIT[4:7] to the package 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.

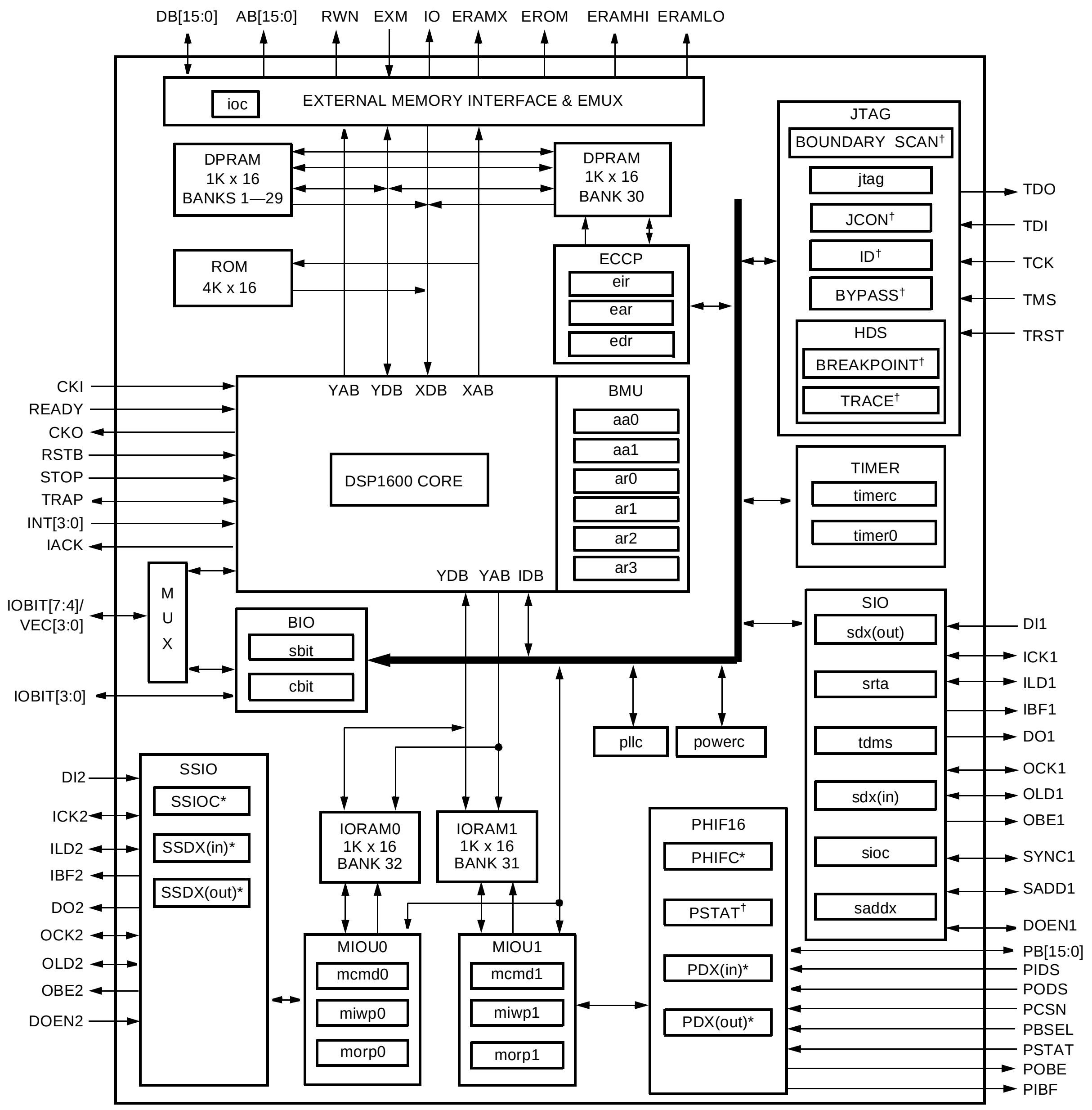
### Power Management

Many applications, such as portable cellular terminals, require programmable sleep modes for power management. There are three different control mechanisms for achieving low-power operation: the **powerc** control register, the STOP pin, and the AWAIT bit in the **alf** register. The **powerc** register configures various power-saving modes by controlling internal clocks and peripheral I/O units. The STOP pin controls the internal processor clock. The AWAIT bit in the **alf** register allows the processor to go into a power-saving standby mode until an interrupt occurs. The various power management options can be chosen based on power consumption and/or wake-up latency requirements.

### Error Correction Coprocessor (ECCP)

The ECCP performs full Viterbi decoding with instructions for MLSE equalization and convolutional decoding. It is designed for 2-tap to 6-tap MLSE equalization with Euclidean branch metrics and rate 1/1 to 1/6 convolutional decoding using constraining lengths from 2 to 7 with Euclidean or Manhattan branch metrics. Two variants of soft-decoded symbols, as well as hard-decoded symbols, can be programmed. The ECCP operates in parallel with the DSP1600 core, increasing the throughput rate. Single instruction Viterbi decoding provides significant code compression required for single DSP solutions in modern digital cellular applications. The ECCP is the source of two interrupts and one flag to the DSP1600 core.

## 4 Hardware Architecture (continued)



* These registers are accessible through the MIOU command registers (mcmd0 and mcmd1).
† These registers are accessible through external pins only.

5-4142(F).e

**Figure 3. DSP1620 Block Diagram**

## 4 Hardware Architecture (continued)

**Table 2. DSP1620 Block Diagram Legend**

| Symbol | Name |
|---|---|
| aa<0—1> | Alternate Accumulators. |
| ar<0—3> | Auxiliary BMU Registers. |
| BIO | Bit I/O Unit. |
| BMU | Bit Manipulation Unit. |
| BREAKPOINT | Four Instruction Breakpoint Registers. |
| BYPASS | JTAG Bypass Register. |
| cbit | BIO Control Register. |
| DPRAM | Dual-Port Random Access Memory. |
| ECCP | Error Correction Coprocessor. |
| ear | ECCP Address Register. |
| edr | ECCP Data Register. |
| eir | ECCP Instruction Register. |
| EMUX | External Memory Multiplexer. |
| HDS | Hardware Development System. |
| ID | JTAG Device Identification Register. |
| IDB | Internal Data Bus. |
| ioc | I/O Configuration Register. |
| IORAM0 | Bank 32 of Internal Data RAM: Shared with MIOU0. |
| IORAM1 | Bank 31 of Internal Data RAM: Shared with MIOU1. |
| JCON | JTAG Configuration Registers. |
| JTAG | JTAG Test Access Port. |
| jtag | 16-bit Serial/Parallel Register. |
| MIOU0 | Modular I/O Unit 0: Controls SSIO. |
| mcmd0 | MIOU0 Command Register. |
| miwp0 | MIOU0 IORAM0 Input Data Write Pointer. |
| morp0 | MIOU0 IORAM0 Output Data Read Pointer. |
| MIOU1 | Modular I/O Unit 1: Controls PHIF16. |
| mcmd1 | MIOU1 Command Register. |
| miwp1 | MIOU1 IORAM1 Input Data Write Pointer. |
| morp1 | MIOU1 IORAM1 Output Data Read Pointer. |
| MUX | Multiplexer. |
| PHIF16 | 16-bit Parallel Host Interface. |
| PDX(in) | PHIF16 Input Data Register. |
| PDX(out) | PHIF16 Output Data Register. |
| PHIFC | Parallel Host Interface Control Register: Programmed Through MIOU1. |
| pllc | Phase-Locked Loop Control Register. |
| powerc | Power Control Register. |
| PSTAT | Parallel Host Interface Status Register. |
| ROM | Internal ROM. |
| saddx | SIO Multiprocessor Protocol Register. |
| sbit | BIO Status Register. |
| sdx(in) | Serial Data Transmit Input Register. |
| sdx(out) | Serial Data Transmit Output Register. |
| SIO | Serial I/O Unit. |
| sioc | Serial I/O Control Register. |
| srta | Serial Receive/Transmit Address Register. |
| SSIO | Simple Serial I/O Unit. |
| SSIOC | Serial I/O Control Register for SSIO: Programmed Through MIOU0. |

## 4 Hardware Architecture (continued)

**Table 2. DSP1620 Block Diagram Legend** (continued)

| Symbol | Name |
|---|---|
| SSDX(in) | I/O Data Input Register. |
| SSDX(out) | I/O Data Output Register. |
| tdms | Serial I/O Time-Division Multiplex Signal Control Register. |
| TIMER | Programmable Timer. |
| timer0 | Timer Running Count Register. |
| timerc | Timer Control Register. |
| TRACE | Program Discontinuity Trace Buffer. |
| XAB | Program Memory Address Bus. |
| XDB | Program Memory Data Bus. |
| YAB | Data Memory Address Bus. |
| YDB | Data Memory Data Bus. |

# 4 Hardware Architecture (continued)

## 4.2 DSP1600 Core Architectural Overview

Figure 4 shows a block diagram of the DSP1600 core.

### System Cache and Control Section (SYS)

This section of the core contains a 15-word cache memory and controls the instruction sequencing. It handles vectored interrupts and traps, and also provides decoding for registers outside of the DSP1600 core. SYS stretches the processor cycle if wait-states are required (wait-states are programmable for external memory accesses). SYS also sequences downloading via JTAG of self-test programs to on-chip dual-port RAM.

The cache loop iteration count can be specified at run time under program control as well as at assembly time.

### Data Arithmetic Unit (DAU)

The data arithmetic unit (DAU) contains a 16 x 16-bit parallel multiplier that generates a full 32-bit product in one instruction cycle. The product can be accumulated with one of two 36-bit accumulators. The accumulator data can be directly loaded from, or stored to memory in two 16-bit words with optional saturation on overflow. The arithmetic logic unit (ALU) supports a full set of arithmetic and logical operations on either 16- or 32-bit data. A standard set of flags can be tested for conditional ALU operations, branches, and subroutine calls. This procedure allows the processor to perform as a powerful 16- or 32-bit microprocessor for logical and control applications. The available instruction set is fully compatible with the DSP1627 instruction set. See Section 5.1 for more information on the instruction set.

The user also has access to two additional DAU registers. The **psw** register contains status information from the DAU (see Table 46, psw — Processor Status Word Register). The arithmetic control register, **auc**, is used to configure some of the features of the DAU (see Table 35) including single-cycle squaring. The **auc** register alignment field supports an arithmetic shift left by one and left or right by two. The **auc** register is cleared by reset.

The counters **c0**, **c1**, and **c2** are signed, 8 bits wide, and are used to count events such as the number of times the program has executed a sequence of code. They are controlled by the conditional instructions and provide a second convenient method of program looping.

### Y Space Address Arithmetic Unit (YAAU)

The YAAU supports high-speed, register-indirect, compound, and direct addressing of data (Y) memory. Four general-purpose 16-bit registers, **r0** to **r3**, are available in the YAAU. These registers can be used to supply the read or write addresses for Y space data. The YAAU also decodes the 16-bit data memory address and outputs individual memory enables for the data access. The YAAU can address the thirty-two 1 Kword banks of on-chip DPRAM/IORAM and a maximum of 128 Kwords of external storage.

Two 16-bit registers, **rb** and **re**, allow zero-overhead modulo addressing of data for efficient filter implementations. Two 16-bit signed registers, **j** and **k**, are used to hold user-defined postmodification increments. (**k** is used only for compound addressing.) Fixed increments of +1, –1, and +2 are also available. Four compound addressing modes are provided to make read/write operations more efficient.

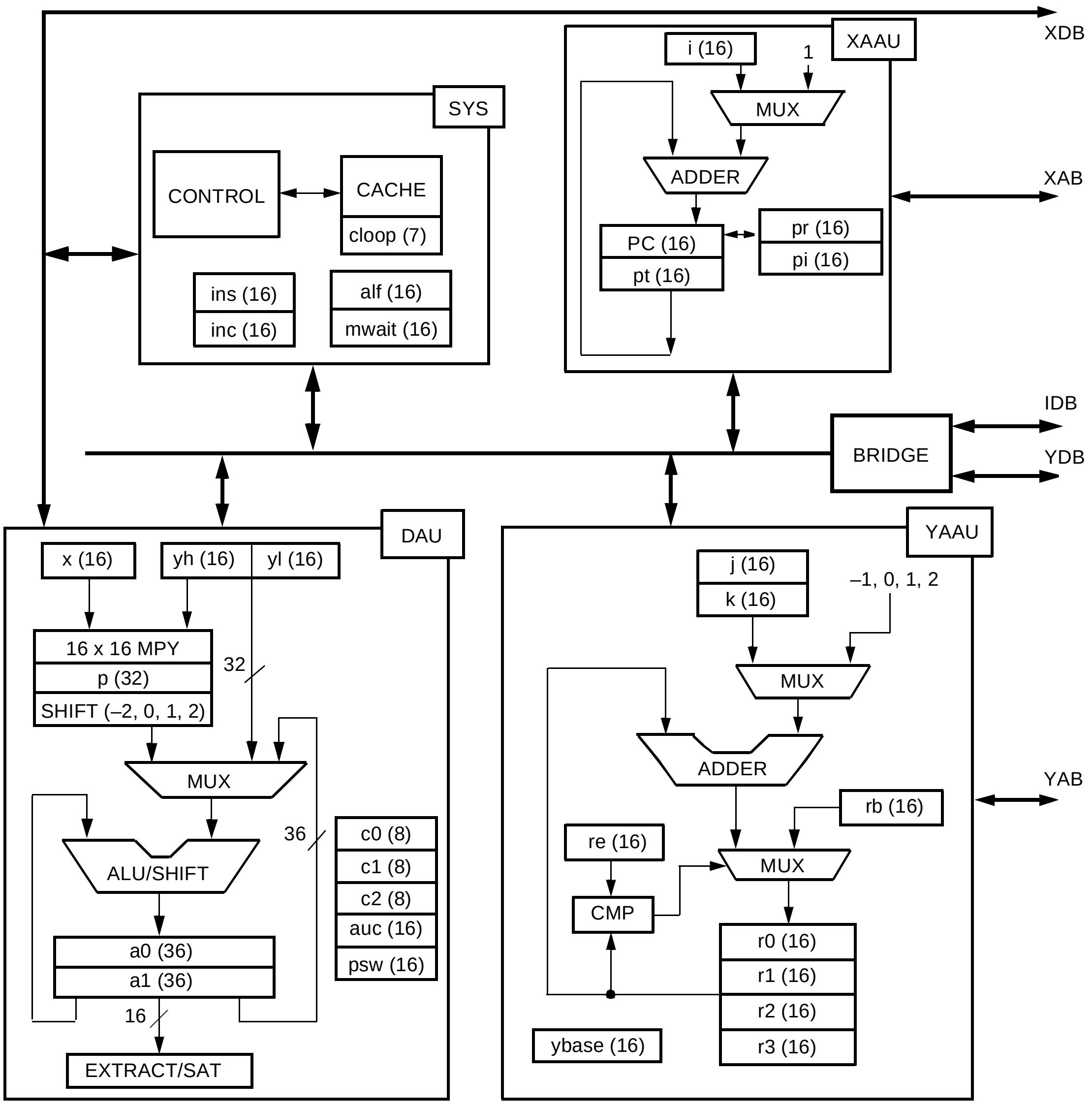
The YAAU allows direct (or indexed) addressing of data memory. In direct addressing, the 16-bit base register (**ybase**) supplies the 11 most significant bits of the address. The direct data instruction supplies the remaining 5 bits to form an address to Y memory space and also specifies one of 16 registers for the source or destination.

### X Space Address Arithmetic Unit (XAAU)

The XAAU supports high-speed, register-indirect, instruction/coefficient memory addressing with postmodification of the register. The 16-bit **pt** register is used for addressing coefficients. The 16-bit signed register **i** holds a user-defined postincrement. A fixed postincrement of +1 is also available. Register PC is the program counter and is not directly accessible by the user. 16-bit registers **pr** and **pi** hold the return address for subroutine calls and interrupts, respectively.

The XAAU decodes the 16-bit instruction/coefficient address and produces enable signals for the appropriate X memory segment. Addressable instruction/coefficient segments include on-chip IROM, 30 Kwords on-chip DPRAM, and 64 Kwords of external storage. The locations of these memory segments depend upon the memory map selected (see Table 5, Instruction/Coefficient Memory Maps).

## 4 Hardware Architecture (continued)



5-1741(F).b

**Figure 4. DSP1600 Core Block Diagram**

## 4 Hardware Architecture (continued)

**Table 3. DSP1600 Core Block Diagram Legend**

| Symbol | Name |
|---|---|
| 16 x 16 MPY | 16-bit x 16-bit Multiplier. |
| a0—a1 | Accumulators 0 and 1 (16-bit halves specified as **a0**, **a0l**, **a1**, and **a1l**)*. |
| alf | AWAIT, LOWPR, Flags. |
| ALU/SHIFT | Arithmetic Logic Unit/Shifter. |
| auc | Arithmetic Unit Control. |
| c0—c2 | Counters 0—2. |
| cloop | Cache Loop Count. |
| CMP | Comparator. |
| DAU | Data Arithmetic Unit. |
| EXTRACT/SAT | Extract/Saturate. |
| i | Increment Register for the X Address Space. |
| IDB | Internal Data Bus. |
| inc | Interrupt Control Register. |
| ins | Interrupt Status Register. |
| j | Increment Register for the Y Address Space. |
| k | Increment Register for the Y Address Space. |
| MUX | Multiplexer. |
| mwait | External Memory Wait-States Register. |
| p | Product Register. |
| PC | Program Counter. |
| pi | Program Interrupt Return Register. |
| pr | Program Return Register. |
| psw | Processor Status Word. |
| pt | X Address Space Pointer. |
| r0—r3 | Y Address Space Pointers. |
| rb | Modulo Addressing Register (begin address). |
| re | Modulo Addressing Register (end address). |
| SYS | System Cache and Control Section. |
| x | Multiplier Input Register. |
| XAAU | X Space Address Arithmetic Unit. |
| XAB | X Space Address Bus. |
| XDB | X Space Data Bus. |
| y | DAU Register. |
| YAAU | Y Space Address Arithmetic Unit. |
| YAB | Y Space Address Bus. |
| YDB | Y Space Data Bus. |
| ybase | Direct Addressing Base Register. |

* F3 ALU instructions with immediates require specifying the high half of the accumulators as **a0h** and **a1h**.

## 4 Hardware Architecture (continued)

### 4.3 Interrupts and Trap

The DSP1620 supports prioritized, vectored interrupts and a trap. The device has eleven internal hardware interrupt sources and four external interrupt pins. Additionally, there is a trap pin and a trap signal from the hardware development system (HDS). A software interrupt is available through the **icall** instruction. The **icall** instruction is reserved for use by the HDS. Each of these sources of interrupt and trap has a unique vector address and priority assigned to it.

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 **inc** register (see Table 39, inc — Interrupt Control Register) and monitored in the **ins** register (see Table 40, ins — Interrupt Status Register). When the DSP1620 goes into an interrupt or trap service routine, the IACK pin is asserted. In addition, pins VEC[3:0] encode which interrupt/trap is being serviced. Table 4 details the encoding used for VEC[3:0].

The DSP1620 WAKEUP interrupt is a new source of core interrupt. WAKEUP is triggered by the logical OR of the PHIF16 input buffer full flag and the SSIO input buffer full flag. The purpose of this interrupt is to reactivate sleeping MIOUs (**alf** AWAIT bit set) and resume peripheral input processing.

#### Interruptibility

Vectored interrupts are serviced only after the execution of an interruptible instruction. If more than one vectored interrupt is asserted at the same time, the interrupts are serviced sequentially according to their assigned priorities. See Table 4 for the priorities assigned to the vectored interrupts. Interrupt service routines, branch and conditional branch instructions, cache loops, and instructions that only decrement one of the RAM pointers, **r0** to **r3** (e.g., ***r3**– –), are not interruptible.

A trap is similar to an interrupt, but it gains control of the processor by branching to the trap service routine even when the current instruction is noninterruptible. It might not be possible to return to normal instruction execution from the trap service routine because the machine state cannot always be saved. In particular, program execution cannot be continued from a trapped cache loop or interrupt service routine. While in a trap service routine, another trap is ignored.

When set to 1, the status bits in the **ins** register indicate that an interrupt has occurred. The processor must reach an interruptible state (completion of an interruptible instruction) before an enabled vectored interrupt is acted on. An interrupt is not serviced if it is not enabled. Polled interrupt service can be implemented by disabling the interrupt in the **inc** register and then polling the **ins** register for the expected event.

#### Vectored Interrupts

Tables 39 and 40 show the **inc** and **ins** registers. A logic 1 written to any bit of **inc** enables (or unmasks) the associated interrupt. If the bit is cleared to a logic 0, the interrupt is masked. Note that neither the software interrupt nor traps can be masked.

The occurrence of an interrupt that is not masked causes the program execution to transfer to the memory location pointed to by that interrupt's vector address, assuming no other interrupt is being serviced (see Table 4). The occurrence of an interrupt that is masked causes no automatic processor action, but sets the corresponding status bit in the **ins** register. If a masked interrupt occurs, it is latched in the **ins** register, but the interrupt is not taken. When unlatched, this latched interrupt initiates automatic processor interrupt action. See the *DSP1620 Digital Signal Processor* Information Manual for a more detailed description of the interrupts.

#### Signaling Interrupt Service Status

Five pins of DSP1620 are devoted to signaling interrupt service status. The IACK pin goes high while any interrupt or user trap is being serviced, and goes low when the ireturn instruction from the service routine is issued. Four pins, VEC[3:0], carry a code indicating which of the interrupts or trap is being serviced. Table 4 contains the encodings used by each interrupt.

Traps due to HDS breakpoints have no effect on either the IACK or VEC[3:0] pins. Instead, they show the interrupt state or interrupt source of the DSP when the trap occurred.

## 4 Hardware Architecture (continued)

**Table 4. Interrupt Vectors**

| Source | Vector | Priority | VEC[3:0] | Issued by |
|---|---|---|---|---|
| No Interrupt | — | — | 0x0 | — |
| Software Interrupt | 0x2 | 1 | 0x1 | icall |
| INT0 | 0x1 | 2 | 0x2 | INT0 pin |
| JINT | 0x42 | 3 | 0x8 | jtag pin |
| INT1 | 0x4 | 4 | 0x9 | INT1 pin |
| INT2 | 0x8 | 5 | 0xa | INT2 pin |
| INT3 | 0xc | 6 | 0xb | INT3 pin |
| TIME | 0x10 | 7 | 0xc | timer |
| MIBF0 | 0x14 | 8 | 0xd | MIOU0 (SSIO) |
| MOBE0 | 0x18 | 9 | 0xe | MIOU0 (SSIO) |
| WAKEUP | 0x1c | 10 | 0x0 | SSIO or PHIF16 input buffer full |
| EREADY | 0x20 | 11 | 0x1 | ECCP ready |
| EOVF | 0x24 | 12 | 0x2 | ECCP overflow |
| IBF | 0x2c | 14 | 0x3 | SIO in |
| OBE | 0x30 | 15 | 0x4 | SIO out |
| MIBF1 | 0x34 | 16 | 0x5 | MIOU1 (PHIF16) |
| MOBE1 | 0x38 | 17 | 0x6 | MIOU1 (PHIF16) |
| TRAP from HDS | 0x3 | 18 | —* | breakpoint, jtag, or pin |
| TRAP from User | 0x46 | 19 = highest | 0x7 | pin |

* Traps due to HDS breakpoints have no effect on VEC[3:0] pins.

### Clearing Interrupts

MIOU-reported SSIO and PHIF16 interrupts (MIBF0, MOBE0, MIBF1, MOBE1) are cleared by writing the reporting MIOU's command register with the appropriate length update command. See Section 4.8.

The SIO interrupts (IBF, OBE) are cleared one instruction cycle after reading or writing, as appropriate, the serial data registers **sdx**(in), and **sdx**(out). To account for this latency, the programmer should ensure that a single instruction (**nop** or other) follows the **sdx** read/write instruction before examining the **ins** register or prior to leaving an interrupt service routine (via **ireturn**). This ensures that stale flags are not read or that an erroneous interrupt is not reported following an ireturn.

The JTAG interrupt (JINT) is cleared by reading the **jtag** register.

Eight of the vectored interrupts can be cleared by writing to the **ins** register. Writing a 1 to the INT0, INT1, INT2, INT3, TIME, EREADY, EOVF, or WAKEUP bits in the **ins** will cause the corresponding interrupt status bit to be cleared to a logic 0. The status bit for these vectored interrupts is also cleared when the ireturn instruction is executed, leaving set any other vectored interrupts that are pending.

## 4 Hardware Architecture (continued)

### Traps

The TRAP pin of the DSP1620 is a bidirectional signal. At reset, it is configured as an input to the processor. Asserting the TRAP pin forces a user trap. Once the trap pin is asserted, it must remain asserted until VEC[3:0] is 0xd (acknowledgment). The trap mechanism is used for two purposes: by an application to rapidly gain control of the processor for asynchronous time-critical event handling (typically for catastrophic error recovery). It is also used by the HDS for breakpointing and gaining control of the processor. Separate vectors are provided for the user trap (0x46) and the HDS trap (0x3). Traps are not maskable.

A trap has four cycles of latency. At most, two instructions execute from the time the trap is received at the pin to when it gains control. An instruction that is executing when a trap occurs is allowed to complete before the trap service routine is entered. (Note that the instruction could be lengthened by wait-states.) During normal program execution, the **pi** register contains either the address of the next instruction (two-cycle instruction executing) or the address following the next instruction (one-cycle instruction executing). In an interrupt service routine, **pi** contains the interrupt return address. When a trap occurs during an interrupt service routine, the value of the **pi** register is overwritten. Specifically, it is not possible to return to an interrupt service routine from a user trap (0x46) service routine. Also, continuing program execution when a trap occurs during a cache loop is not possible.

The HDS trap causes circuitry to force the program memory map to XMAP1 (with on-chip ROM starting at address 0x0) when the trap is taken. The previous memory map is restored when the trap service routine exits by issuing an ireturn. The map is forced to XMAP1 because the HDS code resides in the on-chip ROM.

Using the Lucent Technologies development tools, the TRAP pin can be configured to be an output or an input vectoring to address 0x3. In a multiprocessor environment, the TRAP pins of all the DSPs present can be tied together. During HDS operations, one DSP is selected by the host software to be the master. The master processor's TRAP pin is configured to be an output. The TRAP pins of the slave processors are configured as inputs. When the master processor reaches a breakpoint, the master's TRAP pin is asserted. The slave processors respond to their TRAP input by beginning to execute the HDS code.

### Wait for Interrupt (Standby or Sleep Mode)

The DSP1620 has a power-saving standby mode in which the internal processor 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 operated 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 remains 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.

The AWAIT bit should be set from within the cache if the code that is executing resides in external program memory where more than one wait-state has been programmed. This ensures that an interrupt does not disturb the device from completely entering the sleep state.

For additional power savings, in addition to setting **alf** to the value 0x8000, set **ioc** to the value 0x0180 and **timerc** to the value 0x0040. This holds the CKO pin low and shuts down the timer and prescaler (see Table 54 and Table 41).

Power consumption can be further reduced by by activating other available low-power modes. See Power Management beginning on page 71 for information on these other modes.

## 4 Hardware Architecture (continued)

### 4.4 Memory Maps and Wait-States

The DSP1600 core implements a modified Harvard architecture that has separate on-chip 16-bit address and data buses for the instruction/coefficient (X) and data (Y) memory spaces. The DSP1620 provides a multiplexed external bus that accesses external RAM (ERAM), ROM (EROM), and memory-mapped I/O space (I/O). Programmable wait-states are provided for external accesses.

Both the instruction/coefficient memory space and data memory space are configurable to provide application flexibility.

Table 5 shows the DSP1620 instruction/coefficient memory space maps including the values for the external ROM enable pin (EROM) and address range of the external memory interface address bus (AB).

Table 6 shows the DSP1620 data memory space including values for the EROM, ERAMHI, ERAMLO, ERAMX, IO, and AB external memory interface pins.

#### Instruction/Coefficient Memory Map Selection

Three parameters are used to select the active instruction/coefficient memory map: LOWPR, EXTROM, and EXM.

The LOWPR bit of the **alf** register is initialized to 0 automatically at reset. LOWPR controls the starting address of the thirty 1K banks of DPRAM. If LOWPR is low, DPRAM begins at address 0x8000. If LOWPR is high, DPRAM begins at address 0x0. IROM is not visible when LOWPR is asserted.

When LOWPR is asserted, the EXTROM bit of the **ioc** register determines which 32K segment of a possible 64K EROM physical address space is visible to the programmer. If EXTROM is asserted, physical EROM locations 0x0—0x7FFF are visible in the logical address space 0x8000—0xFFFF. If EXTROM is deasserted, physical EROM locations 0x8000—0xFFFF are visible in the logical address space 0x8000—0xFFFF.

If LOWPR is deasserted, the value of the EXM pin at reset determines whether the internal 4 Kwords ROM (IROM) or EROM locations 0x0—0x7FFF are addressable in the address range 0x0—0x7FFF.

The Lucent Technologies development system tools, together with the on-chip HDS circuitry and the JTAG port, can independently set the memory map. Specifically, during an HDS trap, the memory map is forced to XMAP1. The user's map selection is restored when the trap service routine has completed execution.

## 4 Hardware Architecture (continued)

### XMAP1

XMAP1 has IROM starting at 0x0 and 30 Kwords of DPRAM starting at 0x8000. XMAP1 is established if DSP1620 has EXM low at reset and the LOWPR parameter is programmed to zero. XMAP1 is also used during an HDS trap.

### XMAP2

XMAP2 has 32 Kwords of external ROM (physical EROM addresses 0x0000—0x7FFF) starting at address 0x0. As in XMAP1, 30 Kwords of DPRAM begins at address 0x8000.

### XMAP3

XMAP3 has 30 Kwords of DPRAM starting at address 0x0. 32 Kwords of EROM (physical EROM addresses 0x8000—0xFFFF) storage begins at 0x8000.

### XMAP4

XMAP4 has 30 Kwords of DPRAM starting at address 0x0. 32 Kwords of EROM (physical EROM addresses 0x0000—0x7FFF) storage begins at 0x8000.

**Table 5. Instruction/Coefficient Memory Maps**

| **X Address** | **XMAP1*** <br> **EXM = 0** <br> **EXTROM = X†** <br> **LOWPR = 0‡** | **XMAP2** <br> **EXM = 1** <br> **EXTROM = X†** <br> **LOWPR = 0** | **XMAP3** <br> **EXM = X§** <br> **EXTROM = 0** <br> **LOWPR = 1** | **XMAP4** <br> **EXM = X§** <br> **EXTROM = 1** <br> **LOWPR = 1** |
|---|---|---|---|---|
| 0x0000—0x0FFF | IROM <br> (4K) <br> EROM = 1 | EROM <br> (32K) <br> EROM = 0 <br> AB = 0x0000—0x7FFF | DPRAM <br> (30K) <br> EROM = 1 | DPRAM <br> (30K) <br> EROM = 1 |
| 0x1000—0x77FF | RESERVED | | | |
| 0x7800—0x7FFF | | | RESERVED | RESERVED |
| 0x8000—0xF7FF | DPRAM <br> (30K) <br> EROM = 1 | DPRAM <br> (30K) <br> EROM = 1 | EROM <br> (32K) <br> EROM = 0 <br> AB = 0x8000—0xFFFF | EROM <br> (32K) <br> EROM = 0 <br> AB = 0x0000—0x7FFF |
| 0xF800—0xFFFF | RESERVED | RESERVED | | |

* MAP1 is set automatically during an HDS trap. The user-selected map is restored at the end.
† EXTROM is a don't care when LOWPR is deasserted.
‡ LOWPR is **alf** register bit 14. The Lucent Technologies development system tools can independently set the memory map.
§ EXM is a don't care when LOWPR is asserted.

## 4 Hardware Architecture (continued)

### Boot from External ROM

After RSTB goes from low to high, the DSP1620 comes out of reset and fetches an instruction from address zero of the instruction/coefficient space. The physical location of address zero is determined by the memory map in effect. If EXM is high at the rising edge of RSTB, XMAP2 is selected. XMAP2 has EROM at location zero; thus, program execution begins from external memory. If EXM is high and INT1 is low when RSTB rises, the **mwait** register defaults to 15 wait-states for all external memory segments. If INT1 is high, the **mwait** register defaults to 0 wait-states.

### Data Memory Map Selection

Data memory map selection is based upon the value of the EXTROM and WEROM **ioc** register bits.

#### YMAP1

In YMAP1, the programmer can access 32 Kwords of DPRAM (logical address 0x0000—0x7FFF), and the most significant half of the 64 Kwords physical ERAM space.

#### YMAP2

In YMAP2, the programmer can access 32 Kwords of DPRAM (logical address 0x0000—0x7FFF), and the least significant half of the 64 Kwords physical ERAM space.

#### YMAP3

In YMAP3, the programmer can access 32 Kwords of DPRAM (logical address 0x0000—0x7FFF), and the most significant half of the 64 Kwords physical EROM space.

#### YMAP4

In YMAP4, the programmer can access 32 Kwords of DPRAM (logical address 0x0000—0x7FFF), and the least significant half of the 64 Kwords physical EROM space.

## 4 Hardware Architecture (continued)

**Table 6. Data Memory Maps**

| **Y Address** | **YMAP 1<br>EXTROM = 0<br>WEROM = 0** | **YMAP 2<br>EXTROM = 1<br>WEROM = 0** | **YMAP 3<br>EXTROM = 0<br>WEROM = 1** | **YMAP 4<br>EXTROM = 1<br>WEROM = 1** |
|---|---|---|---|---|
| 0x0000—0x7FFF | DPRAM<br>(32K)<br>EROM = 1<br>ERAMHI = 1<br>ERAMLO = 1<br>ERAMX = 1<br>IO = 1 | DPRAM<br>(32K)<br>EROM = 1<br>ERAMHI = 1<br>ERAMLO = 1<br>ERAMX = 1<br>IO = 1 | DPRAM<br>(32K)<br>EROM = 1<br>ERAMHI = 1<br>ERAMLO = 1<br>ERAMX = 1<br>IO = 1 | DPRAM<br>(32K)<br>EROM = 1<br>ERAMHI = 1<br>ERAMLO = 1<br>ERAMX = 1<br>IO = 1 |
| 0x8000—0x80FF | I/O<br>(0.25K)<br>EROM = 1<br>ERAMHI = 1<br>ERAMLO = 1<br>ERAMX = 0<br>IO = 0<br>AB = 0x8000—<br>0x80FF | I/O<br>(0.25K)<br>EROM = 1<br>ERAMHI = 1<br>ERAMLO = 1<br>ERAMX = 0<br>IO = 0<br>AB = 0x0000—<br>0x00FF | EROM<br>(32K)<br>EROM = 0<br>ERAMHI = 1<br>ERAMLO = 1<br>ERAMX = 1<br>IO = 1<br>AB = 0x8000—<br>0xFFFF | EROM<br>(32K)<br>EROM = 0<br>ERAMHI = 1<br>ERAMLO = 1<br>ERAMX = 1<br>IO = 1<br>AB = 0x0000—<br>0x7FFF |
| 0x8100—0xBFFF | ERAMLO<br>(15.75K)<br>EROM = 1<br>ERAMHI = 1<br>ERAMLO = 0<br>ERAMX = 0<br>IO = 1<br>AB = 0x8100—<br>0xBFFF | ERAMLO<br>(15.75K)<br>EROM = 1<br>ERAMHI = 1<br>ERAMLO = 0<br>ERAMX = 0<br>IO = 1<br>AB = 0x0100—<br>0x3FFF | | |
| 0xC000—0xFFFF | ERAMHI<br>(16K)<br>EROM = 1<br>ERAMHI = 0<br>ERAMLO = 1<br>ERAMX = 0<br>IO = 1<br>AB = 0xC000—<br>0xFFFF | ERAMHI<br>(16K)<br>EROM = 1<br>ERAMHI = 0<br>ERAMLO = 1<br>ERAMX = 0<br>IO = 1<br>AB = 0x4000—<br>0x7FFF | | |

**Wait-States**

The number of wait-states (from 0 to 15) used when accessing each of the four external memory segments (ERAMLO, I/O, ERAMHI, and EROM) is programmable in the **mwait** register (see Table 42). When the program references memory in one of the four external segments, the internal multiplexer is automatically switched to the appropriate set of internal buses, and the associated external enable of ERAMLO, I/O, ERAMHI, or EROM is issued. The external memory cycle is automatically stretched by the number of wait-states configured in the appropriate field of the **mwait** register. When ERAMX is used to enable the single 64K external segment, the ERAMLO, ERAMHI, and I/O fields of **mwait** must be programmed to reflect the segment's wait-state requirement.

# 4 Hardware Architecture (continued)

## 4.5 External Memory Interface (EMI)

The external memory interface supports read/write operations from instruction/coefficient memory, data memory, and memory-mapped I/O devices. The DSP1620 provides a 16-bit external address bus, AB[15:0], and a 16-bit external data bus, DB[15:0]. These buses are multiplexed between the internal buses for the instruction/coefficient memory and the data memory. Five external memory segment enables, ERAMLO, ERAMX, ERAMHI, IO, and EROM, select the external memory segment to be addressed. Table 5 and Table 6 describe the functionality of these bits.

The ERAMHI, ERAMLO, ERAMX, and IO external memory interface pins provide flexibility in mapping the 64 Kwords YMAP1 and YMAP2 external data memory (Y) space. ERAMHI enables a single 32 Kwords physical segment; ERAMLO enables a single 32 Kwords physical segment with two 256 word holes allocated for memory-mapped I/O (I/O). ERAMX enables a single 64 Kwords physical memory segment (composed of the entire address range described by ERAMLO, ERAMHI, and I/O). ERAMHI, ERAMLO, ERAMX, and I/O segments exist only in the Y address space.*

The EROM segment enable maps a single 64 Kwords physical memory segment that can be addressed from either the X or Y address spaces.

Two possible external system configurations are shown in Figure 5 and Figure 6. Figure 5 illustrates a system constructed from 32K x 16 and 64K x 16 SRAM devices. ERAMHI and ERAMLO segments are assigned individual 32K x 16 SRAM chips; the EROM segment is assigned a single 64K x 16 device. 512 ERAMLO locations are dedicated to the 256 device I/O space and cannot be accessed by software.

Figure 6 illustrates a system constructed from 64K x 16 SRAMs. EROM is assigned a single 64K x 16 chip and ERAMX is also assigned a single 64K x 16 device. Since the ERAMX enable encompasses the hardware mapped I/O space, software actions to I/O space are restricted to writing operations only. (Reads to the I/O space will always access ERAMX locations. If the I/O device also attempts to place data on the bus, a conflict will occur.)

Writes to the I/O space also write the corresponding addresses in ERAMX storage; therefore, ERAMX locations that correspond to I/O-space write locations actually used by the software (possibly less than 256 locations) cannot be used for general-purpose data storage. (They will always hold the last value written to their corresponding I/O write address.)

The flexibility provided by the programmable options of the external memory interface (see Table 41 and Table 42) allows the DSP1620 to interface gluelessly with a variety of commercially available memory chips.

Each of the four external memory segments (ERAMLO, I/O, ERAMHI, and EROM) has a number of wait-states that are programmable (from 0 to 15) by writing to the **mwait** register. When the program references memory in one of the four external segments, the internal multiplexer is automatically switched to the appropriate set of internal buses, and the associated external enable of the segment is issued. The external memory cycle is automatically stretched by the number of wait-states in the appropriate field of the **mwait** register.

When ERAMX is used to enable the single 64K external segment, the ERAMLO, ERAMHI, and I/O fields of **mwait** are used to define the memory wait-state requirements for each region of the unified 64K space.

When writing to external memory, the RWN pin goes low for the external cycle. The external data bus, DB[15:0], is driven by the DSP1620 starting halfway through the cycle. The data driven on the external data bus is automatically held after the cycle for one additional clock period unless an external read cycle immediately follows.

* ERAMLO and ERAMHI each map two 16 Kwords logical segments controlled by the EXTROM segment register; ERAMX maps two 32 Kwords logical segments controlled by the EXTROM segment register. EROM maps two 32 Kwords logical segments controlled by the EXTROM segment register.

## 4 Hardware Architecture (continued)

The DSP1620 has one external address bus and one external data bus for both memory spaces. Since some instructions provide the capability of simultaneous access to both X space and Y space, some provision must be made to avoid collisions for external accesses. The DSP1620 has a sequencer that does the external X access first, and then the external Y access, transparently to the programmer. Wait-states are maintained as programmed in the **mwait** register. For example, let two instructions be executed: the first reads a coefficient from EROM and writes data to ERAM; the second reads a coefficient from EROM and reads data from ERAM. The sequencer carries out the following steps at the external memory interface: read EROM, write ERAM, read EROM, and read ERAM. Each step is done in sequential one-instruction cycle steps, assuming zero wait-states are programmed. Note that the number of instruction cycles taken by the two instructions is four. Also, in this case, the write hold time is zero.

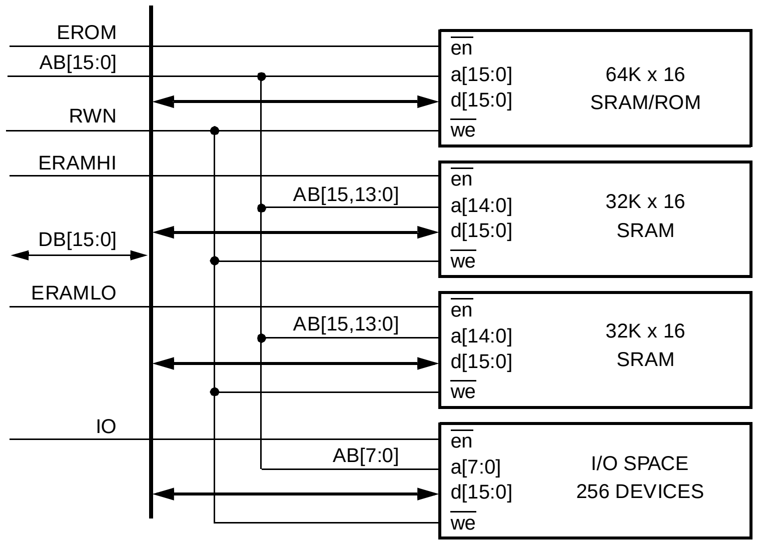
The DSP1620 allows writing into external instruction/coefficient memory: the combination of YMAP3 and YMAP4 provide Y access to the full EROM segment. When accessing EROM from YMAP3 or YMAP4, the I/O, ERAMHI, and ERAMLO **mwait** fields must all be programmed to satisfy the EROM storage's wait-state requirement.

When an access to internal memory is made, the AB[15:0] bus holds the last valid external memory address. Asserting the RSTB pin low 3-states the AB[15:0] bus.

The leading edge of the memory segment enables can be delayed by approximately one-half a CKO period by programming **ioc** register bits DENB[4:0] (see Table 41). This is used to avoid a situation in which two devices drive the data bus simultaneously.

To accommodate both synchronous and asynchronous interfaces, the delay of the falling edge of RWN with respect to the (undelayed) ERAMLO, ERAMHI, ERAMX, and I/O signals can be programmed by the **ioc** register RWNADV bit (see Table 41).

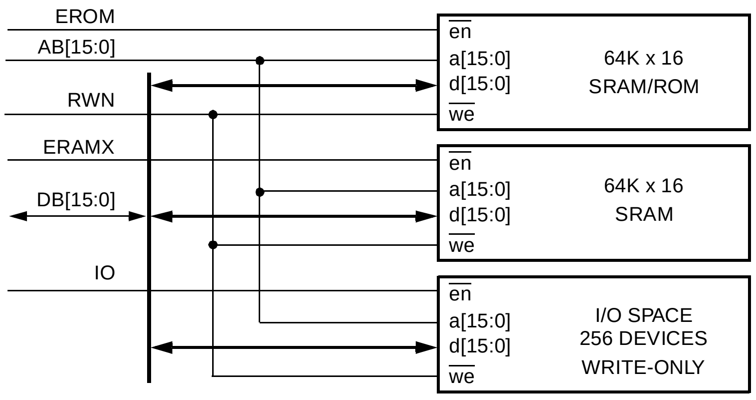
Bits 7, 8, and 13 of the **ioc** register select the mode of operation for the CKO pin (see Table 41). Available options are a free-running unstretched clock, a wait-stated sequenced clock (runs through two complete cycles during a sequenced external memory access), and a wait-stated clock based on the internal instruction cycle. These clocks drop to the low-speed internal ring oscillator when SLOWCKI is enabled. The high-to-low transitions of the wait-stated clock are synchronized to the high-to-low transition of the free-running clock. Also, the CKO pin provides either a continuously high level, a continuously low level, or changes at the rate of the internal processor clock. This last option enables the DSP1620 CKI input buffer to deliver a full-rate clock to other devices while the DSP1620 itself is in one of the low-power modes.



5-4771(F)

**Figure 5. EMI Configuration with 32K x 16 SRAM**

## 4 Hardware Architecture (continued)



5-4772(F)

**Figure 6. EMI Configuration with 64K x 16 SRAM**

## 4 Hardware Architecture (continued)

### READY Pin

The READY input pin permits an external device to extend the length of the EMI access cycle. To extend a DSP's EMI access, an external device must drop the READY pin one CKO clock period ($t_{late}$) plus a setup time ($t_{su}$)* before the programmed wait-states for the DSP's access expire ($t_{wait}$). The width of the READY deassertion pulse is referred to as $t_{rpw}$.

The DSP's access is extended ($t_{ext}$) beyond $t_{wait}$ for at least the number of CKO cycles that READY is deasserted after $t_{late}$ ($t_{stall}$).

- If the READY pin is deasserted at or before $t_{late}$ and held low after $t_{late}$, the DSP's access is extended subject to the following constraint: $t_{stall} \le t_{ext} \le t_{rpw}$.
- If the READY pin is deasserted before $t_{late}$ and subsequently asserted before $t_{late}$, the DSP's access might not be extended: $0 \le t_{ext} \le t_{rpw}$.

If DSP software ensures that back-to-back accesses to the same READY-protected memory region do not occur, the external device can use the low-to-high transition of the associated memory enable (I/O, ERAMHI, ERAMLO, ERAMX, EROM) to signal DSP access completion following reassertion of READY.

The DSP's **mwait** register must be programmed to ensure these timing constraints are satisfied. Let $t_{ready}$ be the external device's CKO-to-READY deassertion time and T be the DSP's clock period; the number of wait-states, W, for the READY-protected region must satisfy the following constraint: $W \ge \text{ceiling}((t_{ready} + t_{su})/T)$.

Figure 7 illustrates the functional behavior of the READY pin for a programmed I/O wait-state value of 2.

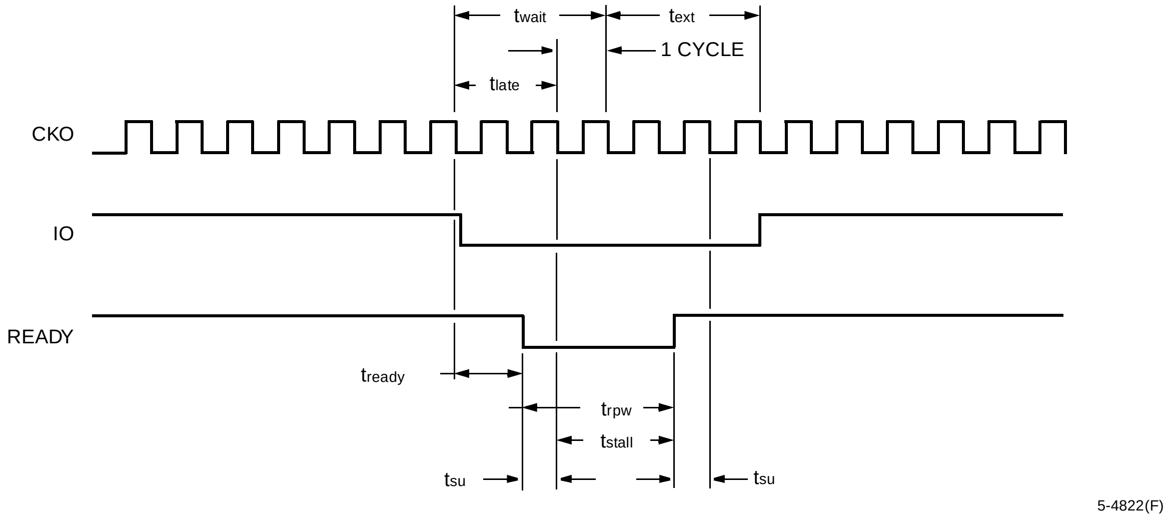


**Figure 7. READY Pin Timing Example for mwait = 0x0020**

* READY setup time ($t_{su}$) is the same as Timing Requirement t140 as shown in Section 10.7 and Section 11.7.

## 4 Hardware Architecture (continued)

### 4.6 Bit Manipulation Unit (BMU)

The BMU interfaces directly to the main accumulators in the DAU providing the following features:

- Barrel shifting—logical and arithmetic, left and right shift
- Normalization and extraction of exponent
- Bit-field extraction and insertion

These features increase the efficiency of the DSP in applications such as control or data encoding and decoding. For example, data packing and unpacking, in which short data words are packed into one 16-bit word for more efficient memory storage, is easily accomplished using the BMU.

In addition, the BMU provides two auxiliary accumulators, **aa0** and **aa1**. In one instruction cycle, 36-bit data can be shuffled, or swapped, between one of the main accumulators and one of the alternate accumulators. The **ar<0—3>** registers are 16-bit registers that control the operations of the BMU. They store a value that determines the amount of shift or the width and offset fields for bit extraction or insertion. Certain operations in the BMU set flags in the DAU **psw** register and the **alf** register (see Table 46 and Table 34). The **ar<0—3>** registers can also be used as general-purpose registers.

The BMU instructions are detailed in Section 5.1. For a thorough description of the BMU, see the *DSP1620 Digital Signal Processor* Information Manual.

### 4.7 Serial I/O Unit (SIO)

The serial I/O port on the DSP1620 device provides a serial interface to many codecs and signal processors with little, if any, external hardware required. The high-speed, double-buffered port (**sdx**) supports back-to-back transmissions of data. The output buffer empty (OBE) and input buffer full (IBF) flags facilitate the reading and/or writing of each serial I/O port by program-driven or interrupt-driven I/O. There are four selectable active clock speeds.

A bit-reversal mode provides compatibility with either the most significant bit (MSB) first or least significant bit (LSB) first serial I/O formats (see Table 50). A multiprocessor I/O configuration is supported. This feature allows up to eight DSP16XX devices to be connected together on an SIO port without requiring external glue logic.

The serial data can be internally looped back by setting the SIO loopback control bit, SIOLBC, of the **ioc** register. SIOLBC affects both the SIO and SSIO. The data output signals are wrapped around internally from the output to the input (DO1 to DI1 and DO2 to DI2). To exercise loopback, the SIO clocks (ICK1, ICK2, OCK1, and OCK2) should either all be in the active mode, 16-bit condition, or each pair should be driven from one external source in passive mode. Similarly, pins ILD1 (ILD2) and OLD1 (OLD2) must both be in active mode or tied together and driven from one external frame clock in passive mode. During loopback, DO1, DO2, DI1, DI2, ICK1, ICK2, OCK1, OCK2, ILD1, ILD2, OLD1, OLD2, SADD1, SYNC1, DOEN1, and DOEN2 are 3-stated.

Setting DODLY = 1 (**sioc**) 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 on OCK.

#### Programmable Modes

Programmable modes of operation for the SIO are controlled by the serial I/O control register (**sioc**). This register, shown in Table 50, 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 DSP1620 generates load and clock signals. When passive, load and clock signal pins are inputs.

Since input and output can be independently configured, the SIO has four different modes of operation. The **sioc** register is also used to select the frequency of active clocks for the SIO. Finally, **sioc** 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 first or LSB first. Input and output data formats can be independently configured.

## 4 Hardware Architecture (continued)

### Multiprocessor Mode

The multiprocessor mode allows up to eight processors to be connected together to provide data transmission among any of the DSPs in the system. The multiprocessor interface is a four-wire interface, consisting of a data channel, an address/protocol channel, a transmit/receive clock, and a sync signal (see Figure 8). The DI1 and DO1 pins of all the DSPs are connected to transmit and receive the data channel. The SADD1 pins of all the DSPs are connected to transmit and receive the address/protocol channel. ICK1 and OCK1 should be tied together and driven from one source. The SYNC1 pins of all the DSPs are connected.

In the configuration shown in Figure 8, the master DSP (DSP 0) generates active SYNC1 and OCK1 signals while the slave DSPs use the SYNC1 and OCK1 signals in passive mode to synchronize operations. In addition, all DSPs must have their ILD1 and OLD1 signals in active mode. ILD1 and OLD1 pins are left open.

While ILD1 and OLD1 are not required externally for multiprocessor operation, they are used internally in the DSP's SIO. Setting the LD1 field of the master's **sioc** register to a logic level 1 ensures that the active generation of SYNC1, ILD1, and OLD1 is derived from OCK1 (see Table 50). With this configuration, all DSPs should use ICK1 (tied to OCK1) in passive mode to avoid conflicts on the clock (CK) line (see the *DSP1620 Digital Signal Processor* Information Manual for more information).

Four registers configure the multiprocessor mode: the time-division multiplex slot register (**tdms**), the serial receive/transmit address register (**srta**), the serial data transmit register (**sdx**), and the multiprocessor serial address/protocol register (**saddx**).

Multiprocessor mode requires no external logic and uses a TDM interface with eight 16-bit time slots per frame. The transmission in any time slot consists of 16 bits of serial data in the data channel and 16 bits of address and protocol information in the address/protocol channel. The address information consists of the transmit address field of the **srta** register of the transmitting device. The address information is transmitted concurrently with the transmission of the first 8 bits of data. The protocol information consists of the transmit protocol field written to the **saddx** register and is transmitted concurrently with the last 8 bits of data (see Table 47, saddx — Multiprocessor Serial/Address Protocol Register.

Data is received or recognized by other DSP(s) whose receive address matches the address in the address/protocol channel. Each SIO port has a user-programmable receive address and transmit address associated with it.

The transmit and receive addresses are programmed in the **srta** register.

In multiprocessor mode, each device can send data in a unique time slot designated by the **tdms** register transmit slot field (bits [7:0]). The **tdms** register has a fully decoded transmit slot field in order to allow one DSP1620 device to transmit in more than one time slot. This procedure is useful for multiprocessor systems with less than eight DSP1620 devices when a higher bandwidth is necessary between certain devices in that system. The DSP operating during time slot 0 also drives SYNC1.

In order to prevent multiple bus drivers, only one DSP must be programmed to transmit in a particular time slot. In addition, it is important to note that the address/protocol channel is 3-stated in any time slot that is not being driven.

To prevent spurious inputs, the address/protocol channel should be pulled up to $V_{DD}$ with a 5 kΩ resistor, or it should be guaranteed that the bus is driven in every time slot. (If the SYNC1 signal is externally generated, then this pull-up is required for correct initialization.)

Each SIO also has a fully decoded transmit address specified by the **srta** register transmit address field (bits [7:0]). This is used to transmit information regarding the destination(s) of the data. The fully decoded receive address specified by the **srta** register receive address field (bits [15:8]) determines which data is received.

The SIO protocol channel data is controlled via the **saddx** register. When the **saddx** register is written, the lower 8 bits contain the 8-bit protocol field. On a read, the high-order 8 bits read from **saddx** are the most recently received protocol field sent from the transmitting DSP's **saddx** output register. The low-order 8 bits are read as 0s.

An example use of the protocol channel is to use the top 3 bits of the **saddx** value as an encoded source address for the DSPs on the multiprocessor bus. This leaves the remaining 5 bits available to convey additional control information, such as whether the associated field is an op code or data, or whether it is the last word in a transfer, etc. These bits can also be used to transfer parity information about the data. Alternatively, the entire field can be used for data transmission, boosting the bandwidth of the port by 50%.

## 4 Hardware Architecture (continued)

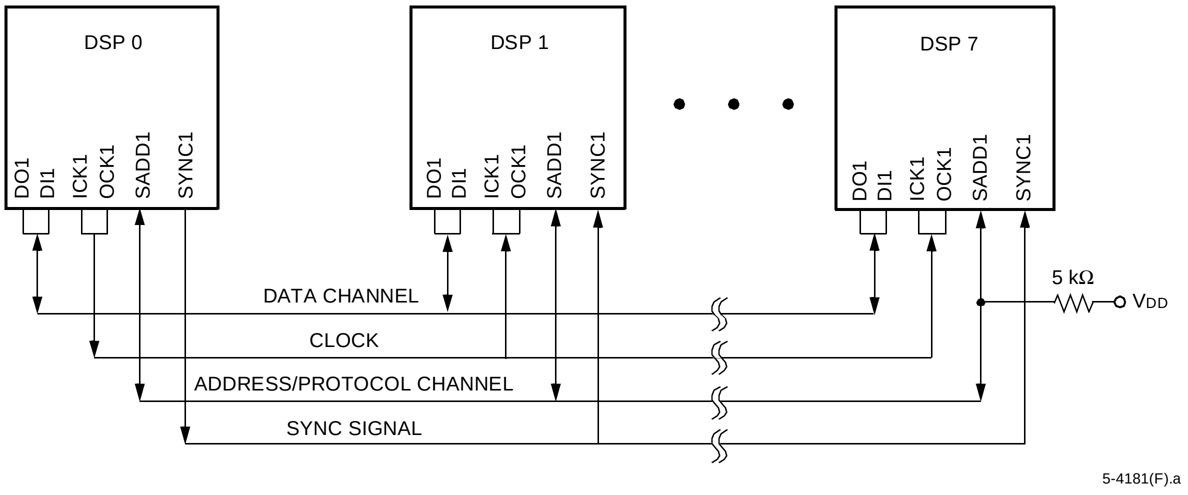


**Figure 8. Multiprocessor Communications and Connections**

# 4 Hardware Architecture (continued)

## 4.8 Modular I/O Unit (MIOU)

The DSP1620 contains two identical modular I/O units: MIOU0 (controls the SSIO) and MIOU1 (controls the PHIF16).

An MIOU provides programmable DMA capability; an MIOU interfaces its attached peripheral to a bank of IORAM storage that resides in the core's Y data memory space. Input and output buffers for the peripheral are allocated in a single 1 Kword bank of IORAM (MIOU0: bank 32, MIOU1: bank 31).

Core hardware supports software transparent DMA access from an MIOU. Concurrent MIOU and core processing are supported. Should core instruction execution and MIOU I/O processing simultaneously require the same IORAM (DPRAM) bank, core execution incurs one wait-state to permit the MIOU access to complete before the core request completes.

MIOU IORAM requests that do not collide with a core IORAM access do not incur a wait-state.

An MIOU remains operational even in low-power mode (clock remains running and is not stopped by AWAIT).

IORAM storage not allocated to an I/O processing area can be used for general-purpose data storage; however, a high core-MIOU IORAM collision rate can impact both core and I/O performance.

An MIOU is disabled when its attached peripheral (PHIF16/SSIO) is disabled by the associated **powerc** register bit (PHIFDIS/SSIODIS). Since **powerc** gates the clock, "clean" reactivation of a powered-off MIOU might not be possible. Therefore, an MIOU should be disabled by the **powerc** register only under the following conditions:

1. The MIOU is not required.
2. The MIOU can be reinitialized by a device reset.

### DSP Configuration for DMA Operation

DMA operations for a port are controlled and configured by manipulating three software visible MIOU registers (*n* = 0: MIOU0; *n* = 1: MIOU1):

- **mcmd*n***: MIOU command register. This software write-only register configures internal MIOU address and control state and provides the means of writing the attached peripheral's control register (**PHIFC** or **SSIOC**).

  MIOU commands are codes consisting of two fields:
  - **mcmd*n***[15:12]: Op code field. Defines the command to be executed.
  - **mcmd*n***[11:0]: Optional parameter field. Data used by the command.
- **miwp*n***: MIOU input write pointer. **miwp*n*** points to the location in the attached IORAM bank to be written with the next input sample (8- or 16-bit) from the attached peripheral; the MIOU advances **miwp*n*** when a sample is written to IORAM. Each input sample consumes one IORAM location. The least significant ten bits contain the IORAM address.
- **morp*n***: MIOU output read pointer. **morp*n*** addresses the IORAM location containing the next output sample to be transferred to the peripheral; the MIOU advances **morp*n*** each time a sample is transferred to the peripheral. The least significant 10 bits contain the IORAM address.

## 4 Hardware Architecture (continued)

MIOU commands are defined in Table 7. MIOU0 commands are issued by writing **mcmd0**; MIOU1 commands are issued by writing **mcmd1.**

**Table 7. MIOU Commands**

| Command Name | Op Code | Parameter | Description |
|---|---|---|---|
| ILEN_UP | 0x4 | Unsigned ILEN update amount.<br>Bit 11 must be 0. | ILEN = ILEN + parameter.<br>Activates peripheral service in an MIOU stalled by a prior RESET command. |
| OLEN_UP | 0x5 | Unsigned OLEN update amount.<br>Bit 11 must be 0. | OLEN = OLEN + parameter |
| IBAS_LD | 0x0 | Physical location in IORAM for base of input area. Bits [11:10] must be 0. | IBAS = parameter |
| OBAS_LD | 0x2 | Physical location in IORAM for base of output area. Bits [11:10] must be 0. | OBAS = parameter |
| ILIM_LD | 0x1 | Physical location in IORAM for limit of input area. Bits [11:10] must be 0. | ILIM = parameter |
| OLIM_LD | 0x3 | Physical location in IORAM for limit of output area. Bits [11:10] must be 0. | OLIM = parameter |
| PCTL_LD | 0x7 | Initialization value for peripheral control registers (SSIOC, PHIFC). | SSIOC/PHIFC = parameter |
| RESET | 0x6 | Must be zero. | Initializes MIOU control state and blocks future MIOU peripheral service until reactivated by subsequent execution of an ILEN_UP command.<br>MIBF: 0<br>MOBE: 1<br>miwp: 0<br>morp: 0<br>OLEN: 0<br>ILEN: –1<br>ILIM, OLIM, OBAS, and IBAS are preserved. |

**mcmd0**, **mcmd1**, **miwp0**, **miwp1**, **morp0**, and **morp1** belong to the core's register set and appear as the destination register of long immediate load instructions (**R = IM16**), load register from Y memory space instructions (**R = Y**), and load register from accumulator instructions (**R = aS[l]**).

In addition, **miwp0**, **miwp1**, **morp0**, and **morp1** appear as the source register in load accumulator from register instructions (**aT[l] = R**) and store to Y memory space instructions (**Y = R**).

## 4 Hardware Architecture (continued)

I/O areas (input and output) are allocated in an MIOU's IORAM physical storage by programming the MIOU's internal base (**IBAS**, **OBAS**) and limit (**OLIM**, **ILIM**) registers. The base registers identify the first physical IORAM location in the (input/output) buffer; the limit registers identify the last physical IORAM location in the (input/output) buffer. MIOU read pointers (**morp0**, **morp1**) and write pointers (**miwp0**, **miwp1**) are circularly advanced within the frame defined by these registers.

A sample code sequence to initialize MIOU0's **IBAS** register follows.

```
#define ibase 0x0100    /*IORAM physical location 0x100*/
#define WRibase 0x0000  /*MIOU command to load IBAS*/
a0 = WRibase | ibase    /*Or in command*/
mcmd0 = a0              /*Load MIOU0's IBAS register*/
```

Similar code sequences are used to initialize an MIOU's **OBAS**, **OLIM**, and **ILIM** registers.

Figure 9 illustrates the IORAM and register relationships for an active MIOU.

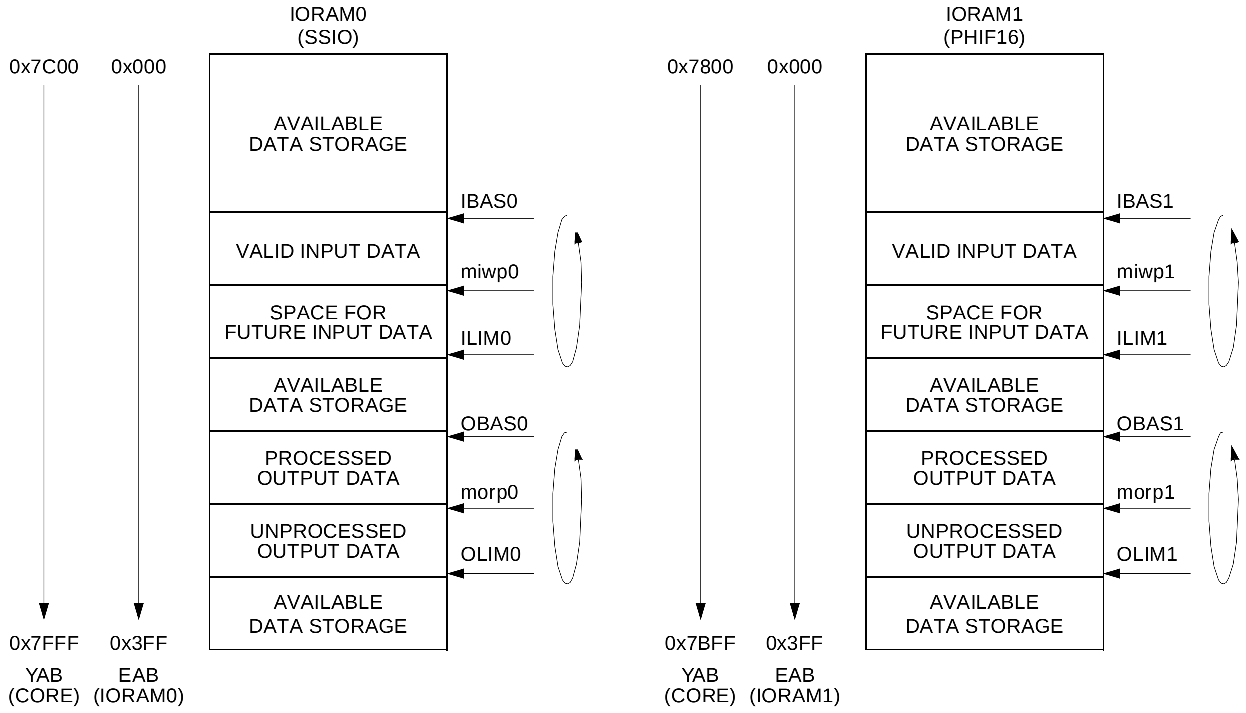


**Figure 9. IORAM Configuration for Active MIOU**

The read/write pointer registers for each area (input/output) use the associated base (**IBAS**/**OBAS**) and limit (**ILIM**/**OLIM**) registers to implement a circular buffer (buffer size = xLIM – xBAS + 1). The register **miwp*n*** increments each time a sample is transferred from the input port to IORAM. When **miwp*n*** equals ILIM, **miwp** is loaded with **IBAS** at the completion of the next input transaction. The register **morp*n*** increments each time a sample is transferred from IORAM to the associated output port. When **morp*n*** equals **OLIM**, **morp*n*** is loaded with **OBAS** at the completion of the next output transaction.

## 4 Hardware Architecture (continued)

### DMA Flow Control

The MIOU permits multiple input samples (up to 1K samples) to be processed without core intervention (buffered I/O). The core and MIOU cooperate to manage the input flow by updating the MIOU internal input length register **ILEN**.

**ILEN** is initialized to 0xFFF (–1) by device reset or MIOU execution of an MIOU reset command. The MIOU uses **ILEN** to assert its input buffer full flag (MIOU0: MIBF0, MIOU1: MIBF1) to the core. Input buffer full is asserted under the following condition:

- **ILEN** decrements to <0x000. (Consequence of MIOU servicing peripheral input port.)

Input buffer full is reset under the following conditions:

- Execution of an MIOU ILEN update command of length L where: (ILEN + L) $\geq$ 0x000.
- Execution of an MIOU reset command.
- Device reset.

Note that MIOU assertion of MIBF does not necessarily imply that all input buffer resources are exhausted (as IBF does for the SIO). MIBF is a flow control signal and does not affect MIOU processing of I/O data. **ILEN**, maintained as a 12-bit two's complement number, is decremented by the MIOU each time it transfers an input sample from the peripheral's input port to IORAM.

The core programs the MIBF trigger depth by issuing an ILEN update command. For example:

```
mcmd0 = 0x4010/*Add 16 to MIOU0's current
                input length count*/
```

The parameter field of the ILEN update command indicates the number of input samples to be processed by the MIOU before it asserts MIBF.

The ILEN update command causes the MIOU to add the current (signed) value of **ILEN** to the (unsigned) lower 10 bits of the command. The sum must be less than 1024. Exceeding this limit results in undefined MIBF behavior; to correct this situation, an MIOU reset command must be issued.

Software must prevent **ILEN** from decrementing below –1023. Exceeding this limit results in undefined MIBF behavior; to correct this situation, an MIOU reset command must be issued.

To ensure software configuration of input control registers is not disturbed by simultaneous MIOU peripheral service operations, execution of an MIOU reset command prevents the MIOU from servicing peripheral requests (input or output) until a subsequent MIOU ILEN update command has been executed. This permits software to establish a consistent **ILEN**-**miwp** register pair.

As a general practice, software should initialize **ILEN** (RESET, . . ., ILEN update) with the logical buffer size (number of samples), L1, of the first input transaction. When indicated by MIBF, software processes the first logical buffer (using L1) and issues an ILEN update command with parameter length equal to the number of samples in the next logical buffer (L2). Since MIOU and core processing are overlapped, this new buffer might already be filled.

The ILEN update 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 reports. If ILEN update operations (L1, L2) are issued without synchronizing with an intervening MIBF, the subsequent MIBF report occurs when the (L1 + L2) samples are processed.

The MIOU permits multiple output samples (up to 1K samples) to be processed without core intervention. The core and MIOU cooperate to manage the output flow control by updating the MIOU internal output length register **OLEN**. The MIOU uses **OLEN** (11-bit, unsigned) to validate the transfer of data from IORAM to the peripheral's output port. While **OLEN** is nonzero, the MIOU transfers I/O samples from IORAM to the peripheral's output port. The MIOU decrements **OLEN** as each sample is transferred.

When **OLEN** decrements to zero, the MIOU ceases output processing and asserts the output buffer empty flag to the core (MIOU0: MOBE0, MIOU1: MOBE1).

Core software initiates (or maintains) MIOU output processing by issuing an OLEN update command to the MIOU. For example:

```
mcmd0= 0x5018/*Add 24 to MIOU0's current
               output length count*/
```

MOBE is reset when the MIOU completes execution of an OLEN update command of length L, where L > 0x000. The OLEN update command causes the MIOU to add the current (unsigned) value of **OLEN** to the (unsigned) lower 10 bits of the command. To prevent undefined behavior, software must ensure the sum is less than or equal to 1024.

The MIOU produces an mioubusy signal that indicates that the MIOU has unfinished output operations pending. When the mioubusy is deasserted (0), all scheduled output transfers are complete, and the DSP can safely enter sleep mode.

The mioubusy signals from both MIOU0 and MIOU1 are ORed together to produce the software visible `mioubusy` condition flag and **alf** register bit. See Table 29 and Table 34.

## 4 Hardware Architecture (continued)

### MIOU Performance

The maximum sustained external interrupt rate (combined input and output) supported by an MIOU is (MIPS/5) interrupts/second.

Since the MIOU operates on the DSP's internal wait-stated clock, MIOU performance is reduced (from the maximum) by DSP wait-states. Such wait-states are caused by the following:

W1. DSP X-Y (same instruction) DPRAM collisions.

W2. DSP-MIOU IORAM collisions.

W3. EMI wait-states.

W4 Externally forced wait-states caused by dropping the READY pin low or assertion of STOP.

W5. Internally forced wait-states caused by use of the AWAIT or NOCK mechanisms (time from external input until hardware resets AWAIT/NOCK).

### MIOU Input Processing Applications

The MIOU is a flexible I/O controller. Four examples of MIOU-based input processing are discussed in the following paragraphs: polled input processing, external interrupt-driven processing, MIOU buffer-driven processing, and lazy input processing.

### MIOU Command Latencies

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 8.

## 4 Hardware Architecture (continued)

**Table 8. MIOU Command Latencies**

| Write Operations | Read Operations | Maximum # of Instructions Until New Value Observed by Read Operation |
|---|---|---|
| `miwp` = <value><br>miwp-write to miwp-read. | `a0 = miwp` | 0 |
| `morp` = <value><br>morp-write to morp-read. | `a0 = morp` | 0 |
| `mcmd` = 0x4<non_zero_length><br>ILEN update to MIBF reset. | `a0 = ins` | 0 |
| `mcmd` = 0x5<non_zero_length><br>OLEN update to MOBE reset. | `a0 = ins` | 0 |
| `mcmd` = 0x5<non_zero_length><br>OLEN update to mioubusy set. | `if mioubusy goto wait` | Four single-cycle instructions are required between an OLEN update and code checking mioubusy for completion of the corresponding output operations. For example:<br>`mcmd = 0x5001`<br>`nop`<br>`nop`<br>`nop`<br>`nop`<br>`busy: if mioubusy goto busy`<br>`done: . . .` |
| `mcmd` = 0x5<non_zero_length><br>OLEN update to mioubusy set. | `a0 = alf` | Four single-cycle instructions are required between an OLEN update and code checking **alf**[9] for completion of the corresponding output operations. For example:<br>`mcmd = 0x5001`<br>`nop`<br>`nop`<br>`nop`<br>`nop`<br>`a0 = alf` |
| `mcmd` = 0x6000<br>MIOU RESET to MOBE set and MIBF reset. | `a0 = ins` | 0 |

## 4 Hardware Architecture (continued)

**Polled Input Processing**

Traditional polled input processing can be implemented using the MIOU. However, this form of I/O processing does not take advantage of the potential (made possible by the MIOU) for overlapped I/O and core execution.

Nevertheless, an example of a simple polling routine that transfers input data from the PHIF16 to the input buffer in IORAM (addressed by **r3**) and finally to the logical array in DPRAM locations addressed by **r0** is shown below.

```
                              /* Simple Polling Input Routine */
#define MIBF1     0x0008      /* ins mask value for MIBF1 */
#define END_RAM   0x7800      /* end address of internal DPRAM */
start_boot:
      auc = 0x0
      mcmd1 = 0x6000    /* RESET MIOU*/
      mcmd1 = 0x7001    /* Init PHIF16:8-bit external,16-bit xfers, INTEL mode */
      mcmd1 = 0x0000    /* Init IORAM Input buffer configuration: IBAS = 0 */
      mcmd1 = 0x13ff    /* Init IORAM Input buffer configuration: ILIM = 1023 */
      mcmd1 = 0x4001    /* Initialize Flow Control: ILEN = 1 */
      rb = 0x7800       /* Circulate r3 in I/O BUFFER rb = base*/
      re = 0x7bff       /* Circular limit*/
      r0 = 0x0000       /* Init logical input array pointer*/
      r3 = 0x7800       /* Init SW IORAM readpointer: Start of IORAM0*/
      y = MIBF1
ram16_dl:               /* PIBF interrupt only comes when */
      a1 = ins          /* 16 bits have been brought in */
      a1 & y
      if eq goto ram16_dl
      mcmd1 = 0x4001    /* Reset MIBF with length = 1 update */
      a0 = *r3++        /* read 1 word from IORAM input buffer */
      *r0++ = a0        /* and write word to DPRAM memory */
      call int_countout /* last location? */
      goto ram16_dl
int_countout:
      a1 = r0           /* if end of iram reached, switch */
      a1h-END_RAM       /* to MAP 3       */
      if eq goto done   /* Done with input Processing.....*/

      return
```

## 4 Hardware Architecture (continued)

### External Interrupt-Driven Input Processing

This method of input processing is synchronized by an external interrupt. The software maintains two (or more) physical arrays of input buffer storage within the input area defined by **IBAS** and **ILIM**. Interrupt service routine (ISR) software swaps the current input array with a future input array by modifying **miwp*n*** to address the base of the future array. This approach supports in-place use of I/O data.

**Note:** The main real-time constraint on this type of software is this: **miwp*n*** must be updated to address the future array before the first sample of future data is loaded into the input port.

Key characteristics of this approach are as follows:

- Software manages two I/O regions (double-buffered) within the input buffer space.
- MIOU transfers input samples from the port to sequential locations (addressed by **miwp*n***) within the input area.
- External device triggers I/O complete interrupt.
- ISR software selects future input array for subsequent input data by writing the IORAM index of the new input array into **miwp*n*** (`miwpn = future_array`) and releases the just completed array for in-place software processing.

### MIOU Buffer-Driven Input Processing

Buffer-driven input processing uses the MIOU input buffer length register (**ILEN**) and associated hardware to trigger input buffer full (MIBF) interrupts that drive the processing of input data and manage input buffer flow control. Buffer-driven input processing is most appropriate when DSP software knows the length of each logical input transaction. Note that the length of each input transaction can be different.

In this scheme, **ILEN** represents the number of input samples expected (by software) in a logical input transaction. It is used by the software to trigger MIBF interrupts when a logical input transaction has completed.

**ILEN** is updated (ILEN_new = ILEN_current + update_value) by the core issued MIOU ILEN update command and decremented by the MIOU as each input sample is transferred to IORAM. The MIOU resolves simultaneous update and decrement hazards. An ILEN update command clears MIBF.

To simplify software address generation and in-place processing of logical input transactions, the input area should be constructed (**IBAS**, **ILIM**) so an integral number of transactions fit exactly into the input area (a logical array should not wrap the frame defined by **IBAS** and **ILIM**). To ensure this happens, software needs to modify ILIM on the last logical transaction of each pass through the input area.

Key characteristics of this approach are as follows:

- Software updates the input length counter (**ILEN**) with the number of input samples in the next logical input transaction. (**Note**: This transaction could have already been transferred to IORAM by the MIOU.)

```
    a0 = LogXactionSize     /*Number of input buffer words processed*/
    y = ILEN_upd            /*ILEN update command*/
    a0 = a0 | y             /*Build command*/
    mcmd = a0               /*Write MIOU command register with ILEN update*/
```

- MIOU transfers input words to sequential IORAM locations within the input area; each transfer decrements **ILEN** and circularly advances **miwp*n***.
- When **ILEN** decrements below zero, the MIOU generates an MIBF interrupt. (Subsequent input transactions continue to decrement **ILEN**, advance **miwp*n***, and write IORAM.)
- If the current input transaction is the last in a cycle through the input area, the ISR (optionally) updates ILIM to ensure that an integral number of transactions exactly fit the input area for the next (possibly already occurring in the MIOU) pass through the input area. The ISR updates **ILEN** with the number of samples in the next logical input transaction.

## 4 Hardware Architecture (continued)

The main real-time constraints on this input processing scheme are as follows:

- ILIM must be updated before the MIOU can write the first word into the stale IORAM region.
- Software use of a completed input array must terminate before the input array is overwritten by a new MIOU input operation. (**Note**: The (possibly) variable size of the input area must be taken into account.)

The buffer-driven interrupt mechanism can also be used to implement a software enforced double-buffered scheme within the input area.

### Lazy Input Processing

This approach to input processing requires neither internal nor external interrupts for synchronization.

- Software polls **miwp*n*** (and possibly reads IORAM) during idle DSP periods to determine if input processing is required (based upon the number of locations between the software read pointer and **miwp*n***).
- No external or MIOU interrupts are used to initiate input processing.
- Software-enforced double-buffering or software (ILIM update)/hardware circular buffering can be used.

### MIOU Output Processing Applications

Examples of MIOU buffer driven and lazy output processing are described below.

### MIOU Buffer-Driven Output Processing

Buffer-driven output processing uses the MIOU's output length register (**OLEN**) and associated hardware to generate output buffer empty (MOBE) interrupts to manage output buffer flow control. The MOBE interrupt can be used to implement a software managed double-buffered I/O scheme where **morp*n*** is updated to address an array of new output data (within the **OBAS**, **OLIM** defined output area).

**OLEN** is established by software, decremented by the MIOU (each time an IORAM word is transferred to the output port), and possibly updated by the software. When **OLEN** is exhausted, the MIOU generates an MOBE interrupt. In buffer-driven software, **OLEN** updates are triggered by MOBE interrupts.

Key characteristics of this approach are given below:

- Software writes output data to sequential locations in the output area of IORAM. To initiate an output operation, software writes the output length register **OLEN** with the number of samples (one sample/one IORAM word) to transfer through the output port. The MIOU adds this update value to the current value of **OLEN**.
- **Note**: In buffer-driven software, **OLEN** should be zero when updated by software.

```
#define wdstodo   0x10      /* Number of output IORAM words to process*/
#define OLEN_upd  0x5000    /* OLEN update command*/
a0 = OLEN_upd | wdstodo     /* Build command*/
mcmd = a0                   /* Write MIOU command register with OLEN update*/
```

- The MIOU transfers `wdstodo` words from IORAM locations addressed by **morp*n*** to the output port (incrementing **morp*n*** and decrementing **OLEN**) until **OLEN** is exhausted. When **OLEN** is exhausted, the MIOU generates an output buffer empty interrupt.
- The software can initiate subsequent MIOU output data transfers from IORAM by updating **OLEN**. An MOBE interrupt report is cleared by an OLEN update operation.

## 4 Hardware Architecture (continued)

### Lazy Output Processing

Lazy output processing is not interrupt-driven. Software writes multiple output transfers to IORAM. The OLEN mechanism is used to initiate (and maintain) MIOU output processing. Each software transfer to IORAM is followed by an OLEN update command of an equal number of words. This means of output processing relies on the circular buffer management and **OLEN** length accumulator hardware.

Software transfers a number of words (`wdstodo`) to the output area of IORAM and signals the availability of this data for output processing by updating the **OLEN** register. The MIOU adds `wdstodo` to the current value of **OLEN**.

Note that software issues OLEN update commands to an MIOU even if the MIOU has not finished processing a prior **OLEN**-specified output transaction. The accumulate function of the **OLEN** (and **ILEN**) update command permits overlapping core and MIOU I/O processing. IORAM storage buffers the short-term difference in core/MIOU processing rates.

Software addressing of the IORAM must take into account the circular nature of the output region (if circular wrap is permitted).

The MIOU transfers data from IORAM to the output port whenever **OLEN** is greater than zero.

Output buffer flow control is managed either by the nature of the algorithm (e.g., output resources can be computed from the input demand) or a software comparison of the MIOU output buffer read pointer (**morp*n***) with the software-maintained output buffer write pointer.

## 4.9 Simple Serial I/O Unit (SSIO)

The SSIO port on the DSP1620 device provides a serial interface to many codecs and signal processors with little, if any, external hardware required. The high-speed, double-buffered port supports back-to-back transmissions of data. The SSIO is a DMA peripheral that interfaces to IORAM0 (bank 32) through MIOU0.

There are four selectable active clock speeds.

A bit-reversal mode provides compatibility with either the most significant bit (MSB) first or least significant bit (LSB) first serial I/O formats (see Table 50).

The serial data can be internally looped back by setting the SIO loopback control bit, SIOLBC, of the **ioc** register. SIOLBC affects both the SIO and SSIO.

Setting DODLY = 1 (**SSIOC**) 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 on OCK.

### Programmable Modes

The simple serial I/O control register (**SSIOC**) controls the programmable modes of operation for the SSIO. This register, shown in Table 50, 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 DSP1620 generates load and clock signals. When passive, load and clock signal pins are inputs.

Since input and output can be independently configured, the SIO 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 first or LSB first. Input and output data formats can be independently configured.

The **SSIOC** register is programmed through MIOU0.

# 4 Hardware Architecture (continued)

## 4.10 Parallel Host Interface (PHIF16)

The DSP1620 has an 16-bit parallel host bus interface for rapid transfer of data with external devices. PHIF16 is a DMA peripheral that interfaces to IORAM1, bank 31, through MIOU1.

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- or 16-bit external interface.

The PHIF16 in 8-bit external configuration operates as a DSP1627 PHIF. In 8-bit external configuration, PHIF16 provides for 8-bit or 16-bit logical data transfers. 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 buffer, **PDX**(in), and a 16-bit output buffer, **PDX**(out). **PDX**(in) is loaded with host data from the 16-bit data bus PB[15:0]. **PDX**(out) is loaded by MIOU1 with output data from the IORAM1 location addressed by the **morp1** register. The **PDX** register is not directly accessible by the user.

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 43) and the PHIF16 status register (**PSTAT**, see Table 11). 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 MIOU1 using the peripheral control load command (see Table 7).

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 and enables PIDS and PODS (or PRWN and PDS, depending on the protocol used). While PCSN is high, the DSP1620 ignores any activity on PIDS and/or PODS. If a DSP1620 is intended to be continuously accessed through the PHIF16 port, PCSN should be grounded.

If PCSN is low, the assertion of PIDS and PODS by an external device causes the PHIF16 to recognize a host request. If MIOU1 has been properly programmed, it responds to the host request by either filling **PDX**(out) or emptying **PDX**(in).

### Programmability

The PHIF16 external interface is configured for 8- 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 (MIOU1) 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 9 summarizes the port's output functionality as controlled by the PSTAT and PBSEL pins and the PBSELF and PMODE fields. Table 10 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 are 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 MIOU1.

## 4 Hardware Architecture (continued)

**Table 9. 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 10. 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 11. PSTAT Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | | | PIBF | POBE |

## 4 Hardware Architecture (continued)

### 4.11 Bit I/O 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 48) 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 49) 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 12 shows the functionality of the MODE/MASK and DATA/PAT bits based on the direction selected for the associated IOBIT pin.

Those bits 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). These flags can be used for conditional branch or special instructions. The state of these flags can be saved and restored by reading and writing bits 0 to 3 of the **alf** register (see Table 34).

**Table 12. 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 |

* $0 \le n \le 7$.

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.

**Pin Multiplexing**

Refer to Pin Multiplexing in Section 4.1 for a description of the pin multiplexing of the IOBIT[7:4] and VEC[0:3] pins.

### 4.12 Timer

The interrupt timer is composed of the **timerc** (control) register, the **timer0** register, the prescaler, and the counter itself. The timer control register (see Table 54) sets up the operational state of the timer and prescaler. The **timer0** register is used to hold the counter reload value (or period register) and to set the initial value of the counter. The prescaler slows the clock to the timer by a number of binary divisors to allow for a wide range of interrupt delay periods.

The counter is a 16-bit down counter that can be loaded with an arbitrary number from software. It counts down to 0 at the clock rate provided by the prescaler. Upon reaching 0 count, a vectored interrupt to program address 0x10 is issued to the DSP1620, providing the interrupt is enabled (bit 8 of **inc** and **ins** registers). The counter then either waits in an inactive state for another command from software, or automatically repeats the last interrupting period, depending upon the state of the RELOAD bit in the **timerc** register.

When RELOAD is 0, the counter counts down from its initial value to 0, interrupts the DSP1620, and then stops, remaining inactive until another value is written to the **timer0** register. Writing to the **timer0** register causes both the counter and the period register to be written with the specified 16-bit number. When RELOAD is 1, the counter counts down from its initial value to 0, interrupts the DSP1620, automatically reloads the specified initial value from the period register into the counter, and repeats indefinitely. This provides for either a single timed interrupt event or a regular interrupt clock of arbitrary period.

# 4 Hardware Architecture (continued)

The timer can be stopped and started by software, and can be reloaded with a new period at any time. Its count value, at the time of the read, can also be read by software. Due to pipeline stages, stopping and starting the timer can result in one inaccurate count or prescaled period. When the DSP1620 is reset, the bottom 6 bits of the **timerc** register and the **timer0** register and counter are initialized to 0. This sets the prescaler to CKO/2*, turns off the reload feature, disables timer counting, and initializes the timer to its inactive state. The act of resetting the chip does not cause a timer interrupt. Note that the period register is not initialized on reset.

The T0EN bit of the **timerc** register enables the clock to the timer. When T0EN is a 1, the timer counts down towards 0. When T0EN is a 0, the timer holds its current count.

The PRESCALE field of the **timerc** register selects one of 16 possible clock rates for the timer input clock (see Table 54).

Setting the DISABLE bit of the **timerc** register to a logic 1 shuts down the timer and the prescaler for power savings. Setting the TIMERDIS, bit 4, in the **powerc** register has the same effect as shutting down the timer. The DISABLE bit and the TIMERDIS bit are cleared by writing a 0 to their respective registers to restore the normal operating mode.

## 4.13 Error Correction Coprocessor (ECCP)

The error correction coprocessor (ECCP) performs full Viterbi decoding with single instructions for maximum likelihood sequence estimation (MLSE) equalization and convolutional decoding. The ECCP operates in parallel with the DSP core, increasing the throughput rate, and single-instruction Viterbi decoding provides significant code compression required for a single DSP solution for modern digital cellular applications.

### System Description

The ECCP is a loosely coupled, programmable, internal coprocessor that operates in parallel with the DSP1600 core. A complete Viterbi decoding for MLSE equalization or convolutional decoding is performed with a single DSP instruction.

The core communicates with the ECCP module via three interface registers. The ECCP address register, **ear**, is used to indirectly access the ECCP internal memory-mapped registers. The ECCP data register, **edr**, works in concert with the address register to indirectly read from or write to an ECCP internal memory-mapped register addressed by the contents of the address register. After each **edr** access, the contents of the address register are postincremented by one. Upon writing an ECCP op code to instruction register **eir**, either MLSE equalization, convolutional decoding, a simple traceback operation, or ECCP reset is invoked.

The mode of operation of the ECCP is set up by writing appropriate fields of a memory-mapped control register. In MLSE equalization, the control register can be configured for 2-tap to 6-tap equalization. In convolutional decoding, the control register can be configured for constraint lengths 2 through 7 and code rates 1/1 through 1/6.

One of two variants of the soft-decoded output can be programmed, or a hard-decoded output can be chosen.

Usually, convolutional decoding is performed after MLSE equalization. For receiver configuration with MLSE equalization followed by convolutional decoding, a Manhattan branch metric computation for convolutional decoding can be selected by setting a branch metric select bit in the control register.

In wideband low data rate applications, additive white Gaussian noise (AWGN) is the principle channel impairment, and Euclidean branch metric computation for convolutional decoding is selected by resetting the branch metric select bit to zero.

A traceback length register is provided for programming the traceback decode length.

* Frequency of CKO/2 is equivalent to either CKI/2 for the PLL bypassed or related to CKI by the PLL multiplying factors. See Section 4.16, Clock Synthesis.

## 4 Hardware Architecture (continued)

A block diagram of the coprocessor and its interface to the DSP1600 core is shown in the following figure:

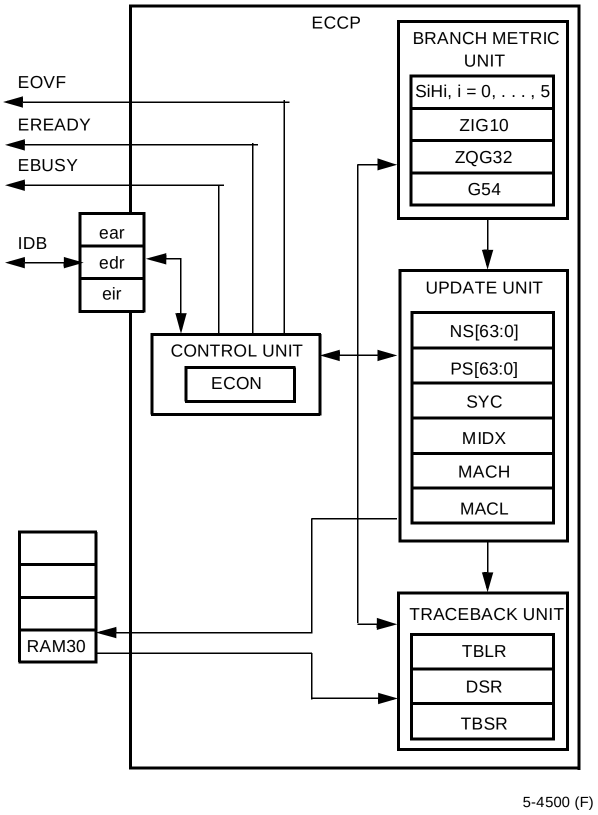


**Figure 10. Error Correction Coprocessor Block Diagram/Programming Model**

The ECCP internal registers are accessed indirectly through the address and data registers, **ear** and **edr**. The control register, ECON, and the traceback length register, TBLR, are used to program the operating mode of the ECCP. The symbol registers (S0H0—S5H5, ZIG10, ZQG32), the generating polynomial registers (ZIG10, ZQG32, G54), and the channel impulse registers (S0H0—S5H5) are used as input to the ECCP for MLSE or convolutional decoding. Following a Viterbi decoding operation, the decoded symbol is read out of the decoded symbol register, DSR. All internal states of these memory-mapped registers are accessible and controllable by the DSP program. During periods of simultaneous DSP-ECCP activity, however, ECCP internal **edr** registers as well as the shared bank of RAM, RAM30, are not accessible to the user's DSP code.

**Branch Metric Unit:** The branch metric unit of the ECCP performs full-precision real and complex arithmetic for computing 16-bit incremental branch metrics required for MLSE equalization and convolutional decoding.

**MLSE Branch Metric:** To generate the estimated received complex signal at instance n, E(n, **k**) = EI(n, **k**) + **j** EQ(n, **k**), at the receiver, all possible states, **k** = 0 to $2^{C-1} - 1$, in the Viterbi state transition are convolved with the estimated channel impulse response, $H(n) = [h(n), h(n-1), h(n-2), \ldots, h(n-C+1)]^T$, where the constraint length C = {2 to 6}. Each in-phase and quadrature-phase part of the channel tap, h(n) = hI(n) + **j** hQ(n), is quantized to an 8-bit 2's complement number. The channel estimates are normalized prior to loading into the ECCP such that the worst-case summation of the hI(n) or hQ(n) are confined within a 10-bit 2's complement number. The in-phase and quadrature-phase parts of the received complex signal Z(n) = ZI(n) + **j** ZQ(n) are also confined within a 10-bit 2's complement number.

The Euclidean branch metric associated with each of the $2^C$ state transitions is calculated as:

$$BM(n, \mathbf{k}) = XI(n, \mathbf{k})^2 + XQ(n, \mathbf{k})^2$$

where

$$XI(n, \mathbf{k}) = abs\{ZI(n) - EI(n, \mathbf{k})\}$$

and

$$XQ = abs\{ZQ(n) - EQ(n, \mathbf{k})\}$$

The absolute values of the difference signal are saturated at level 0xFF. The 16 most significant bits of this 17-bit incremental branch metric are retained for the add-compare-select operation of the Viterbi algorithm.

The in-phase and quadrature-phase parts of the received complex signal are stored in ZIG10 and ZQG32 registers, respectively. The complex estimated channel taps H5 through H0 are stored in S5H5 through S0H0 registers, such that the in-phase part of the channel occupies the upper byte and the quadrature-phase part of the channel occupies the lower byte.

**Convolutional Branch Metric:** Two types of distance computation are implemented for convolutional decoding. Convolutional decoding over a Gaussian channel is supported with Euclidean distance measure for rate 1/1 and 1/2 convolutional encoding. Convolutional decoding preceded by the MLSE equalization or other linear/nonlinear equalization is supported with Manhattan distance measure for rate 1/1 through 1/6 convolutional encoding.

## 4 Hardware Architecture (continued)

Generating polynomials, G(0), . . . , G(5), up to six delays corresponding to a constraint length of seven, take part in computing the estimated received signals, E(0, **k**), . . . , E(5, **k**), within the ECCP associated with all possible state transitions, **k** = 0, 1, $2^{C-1}$.

Six 8-bit soft symbols, S(0), . . . , S(5), are loaded into the ECCP. The incremental branch metrics associated with all $2^C$ state transitions are calculated as indicated in Table 13.

**Table 13. Incremental Branch Metrics**

| Distance Measure | Code Rate | 16-bit Incremental Branch Metric |
|---|---|---|
| Euclidean | 1/1 | $(S(0) - E(0))^2$ |
| Euclidean | 1/2 | $[\Sigma(S(\mathbf{i}) - E(\mathbf{i}))^2] >> 1$, **i** = 0, 1 |
| Manhattan | 1/1 | $[S(\mathbf{i}) - E(\mathbf{i})] << 8$, **i** = 0 |
| Manhattan | 1/2 | $[(S(\mathbf{i}) - E(\mathbf{i}))] << 7$, **i** = 0, 1 |
| Manhattan | 1/3 or 1/4 | $[(S(\mathbf{i}) - E(\mathbf{i}))] << 6$, **i** = 0, 1, 2, or 3 |
| Manhattan | 1/5 or 1/6 | $[(S(\mathbf{i}) - E(\mathbf{i}))] << 5$, **i** = 0, 1, . . . , 4, or 5 |

The received 8-bit signals S(5) through S(0) are stored in the S5H5 through S0H0 registers. The generating polynomials G(1) and G(0) are stored in the upper and lower bytes of the ZIG10 register, respectively. The generating polynomials G(3) and G(2) are stored in the upper and lower bytes of the ZQG32 register, respectively. The generating polynomials G(5) and G(4) are stored in the upper and lower bytes of the G54 register, respectively.

**Update Unit:** The add-compare-select operation of the Viterbi algorithm is performed in this unit. At every time instant, there are $2^C$ state transitions of which $2^{C-1}$ state transitions survive. The update unit selects and updates $2^{C-1}$ surviving sequences in the traceback RAM that consists of the thirtieth bank of the internal RAM, RAM30. The accumulated cost of the path p at the Jth instant, ACC(J, p), is the sum of the incremental branch metrics belonging to the path p up to the time instant J:

$$ACC(J, p) = \Sigma BM(\mathbf{j}, p),\ \mathbf{j} = 1, \ldots, J$$

The update unit computes and stores full-precision 24-bit resolution path metrics of the bit sequence. To assist the detection of a near overflow in the accumulated path cost, an internal vectored interrupt, EOVF, is provided.

**Traceback Unit:** The traceback unit selects a path with the smallest path metric among $2^{C-1}$ survivor paths at every instant. The last signal of the path corresponding to the maximum likelihood sequence is delivered to the decoder output. The depth of this last signal is programmable at the symbol rate. The traceback decoding starts from the minimum cost index associated with the state with the minimum cost, min $\{Acc(\mathbf{j}, p^1), \ldots, Acc(\mathbf{j}, p^{2^{C-1}})\}$. If the end state is known, the traceback decoding can be forced in the direction of the right path by writing the desired end state into the minimum cost index register, MIDX.

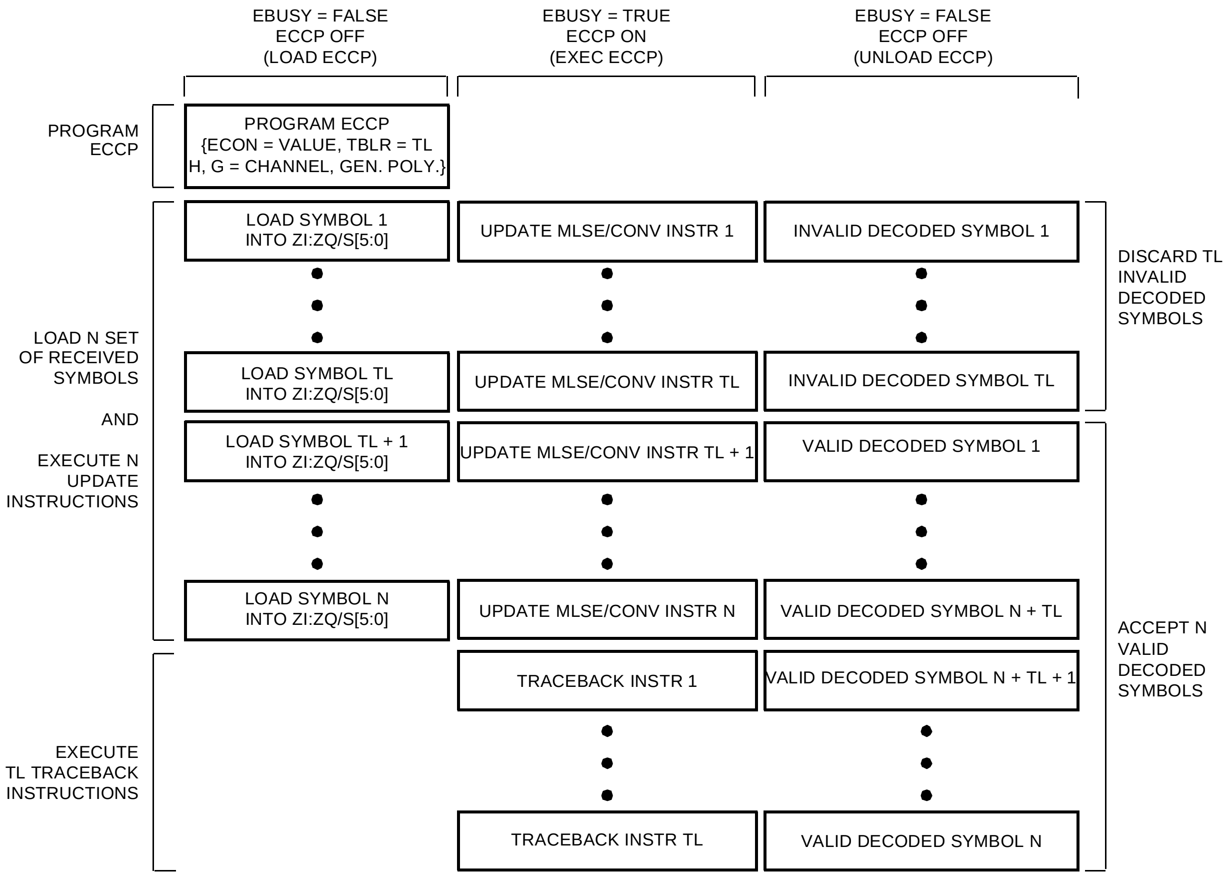
**Traceback RAM**: The thirtieth 1 Kword bank of dual-port RAM is shared between the DSP1600 core and the ECCP. RAM30, located in the Y memory space in the address range 0x7400 to 0x77FF, is used by the ECCP for storing traceback information. When the ECCP is active, i.e., the EBUSY flag is asserted, the DSP core cannot access this traceback RAM.

**Interrupts and Flags:** The ECCP interrupts the DSP1600 core when the ECCP has completed an instruction, EREADY, or when an overflow in the accumulated cost is imminent, EOVF. Also, an EBUSY flag is provided to the core to indicate when the ECCP is in operation.

## 4 Hardware Architecture (continued)

### DSP Decoding Operation Sequence

The DSP operation sequence for invoking the ECCP for an MLSE equalization or convolutional decoding operation is explained with the operation flow diagram in Figure 11.


EBUSY = FALSE
ECCP OFF
(LOAD ECCP)
EBUSY = TRUE
ECCP ON
(EXEC ECCP)
EBUSY = FALSE
ECCP OFF
(UNLOAD ECCP)
PROGRAM ECCP
PROGRAM ECCP {ECON = VALUE, TBLR = TL H, G = CHANNEL, GEN. POLY.}
LOAD SYMBOL 1 INTO ZI:ZQ/S[5:0]
UPDATE MLSE/CONV INSTR 1
INVALID DECODED SYMBOL 1
DISCARD TL INVALID DECODED SYMBOLS
LOAD N SET OF RECEIVED SYMBOLS
AND
EXECUTE N UPDATE INSTRUCTIONS
LOAD SYMBOL TL INTO ZI:ZQ/S[5:0]
UPDATE MLSE/CONV INSTR TL
INVALID DECODED SYMBOL TL
LOAD SYMBOL TL + 1 INTO ZI:ZQ/S[5:0]
UPDATE MLSE/CONV INSTR TL + 1
VALID DECODED SYMBOL 1
LOAD SYMBOL N INTO ZI:ZQ/S[5:0]
UPDATE MLSE/CONV INSTR N
VALID DECODED SYMBOL N + TL
ACCEPT N VALID DECODED SYMBOLS
TRACEBACK INSTR 1
VALID DECODED SYMBOL N + TL + 1
EXECUTE TL TRACEBACK INSTRUCTIONS
TRACEBACK INSTR TL
VALID DECODED SYMBOL N

5-4501(F).a

**Figure 11. DSP Core Operation Sequence**

## 4 Hardware Architecture (continued)

### Operation of the ECCP

To operate the ECCP, the mode of operation is first programmed by setting the control register, ECON, and the traceback length register, TBLR, and appropriately initializing the present state accumulated costs. The complete Viterbi decoding operation is achieved by recursively loading the received symbols into the ECCP, executing the ECCP with an UpdateMLSE, an UpdateConv, or a TraceBack instruction, and unloading the decoded symbol from the ECCP. The operation of the ECCP is captured in the signal flow diagram in Figure 12.

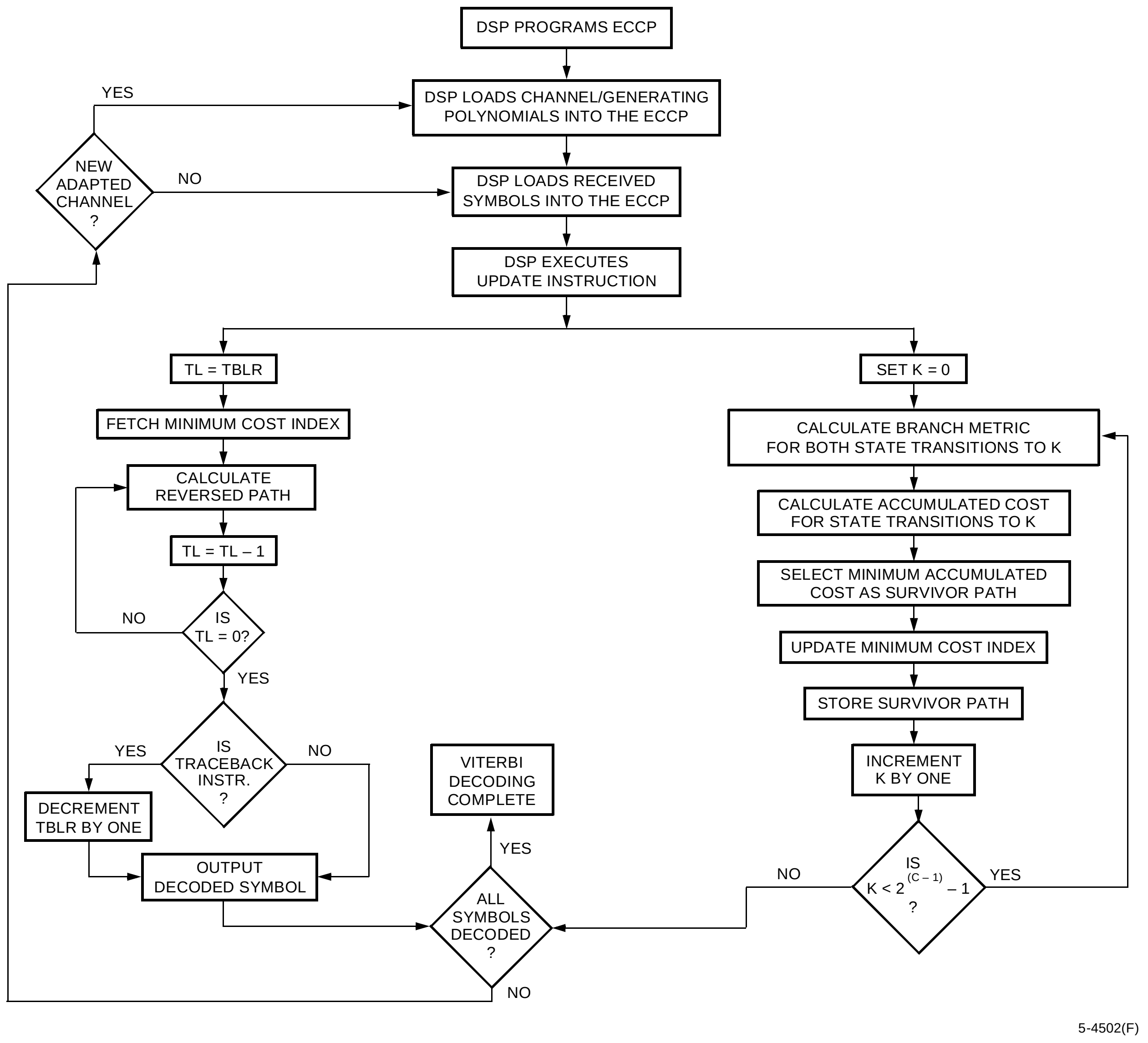

DSP PROGRAMS ECCP
DSP LOADS CHANNEL/GENERATING POLYNOMIALS INTO THE ECCP
YES
NEW ADAPTED CHANNEL ?
NO
DSP LOADS RECEIVED SYMBOLS INTO THE ECCP
DSP EXECUTES UPDATE INSTRUCTION
TL = TBLR
FETCH MINIMUM COST INDEX
CALCULATE REVERSED PATH
TL = TL – 1
NO
IS TL = 0?
YES
YES
IS TRACEBACK INSTR. ?
NO
DECREMENT TBLR BY ONE
OUTPUT DECODED SYMBOL
VITERBI DECODING COMPLETE
YES
ALL SYMBOLS DECODED ?
NO
SET K = 0
CALCULATE BRANCH METRIC FOR BOTH STATE TRANSITIONS TO K
CALCULATE ACCUMULATED COST FOR STATE TRANSITIONS TO K
SELECT MINIMUM ACCUMULATED COST AS SURVIVOR PATH
UPDATE MINIMUM COST INDEX
STORE SURVIVOR PATH
INCREMENT K BY ONE
NO
IS K < 2^(C – 1) – 1 ?
YES
5-4502(F)

**Figure 12. ECCP Operation Sequence**

## 4 Hardware Architecture (continued)

### Software Architecture

The ECCP registers are grouped into two categories: the R-field registers and the internal memory-mapped registers.

**R-Field Registers:** Three registers (**ear**, **edr**, and **eir**) are defined in the core instruction set as programmable registers for executing the ECCP and establishing the data interface between the ECCP and the core. Reserved bits are always zero when read. To make the program compatible with future chip revisions, write the reserved bits with zeros.

**Address Register (ear):** The address register holds the address of the ECCP internal memory-mapped registers. Each time the core accesses an internal ECCP register through **edr**, the content of the address register is postincremented by one. During a DSP compound addressing instruction, the same **edr** register is accessed for both the read and the write operation.

**Data Register (edr):** The contents of the ECCP internal memory-mapped registers are indirectly accessed by the DSP through this register. A write to the data register is directed to the ECCP internal register addressed by the contents of the address register. A read from the data register fetches the contents of the ECCP internal register addressed by the address register. Every access to the **edr** autoincrements the address register, **ear**.

**Instruction Register (eir):** Four instructions are defined for the ECCP operation. These instructions are executed upon writing appropriate values in the **eir** register. Table 14 indicates the instruction encoding and their mnemonics.

**Table 14. ECCP Instruction Encoding**

| eir Value in Hex | Instruction |
|---|---|
| 0000 | UpdateMLSE |
| 0001 | UpdateConv |
| 0002 | TraceBack |
| 0003 | Reserved |
| 0004 | ResetECCP |
| 0005—FFFF | Reserved |

The UpdateMLSE instruction and the UpdateConv instruction each perform an appropriate branch metric calculation, a complete Viterbi add-compare-select operation, and a concurrent traceback decoding operation. The TraceBack instruction performs the traceback decoding alone.

The ResetECCP instruction performs a proper reset operation to initialize various registers as described in Table 15.

**Table 15. Reset State of ECCP Registers**

| Register | Reset State |
|---|---|
| eir | 0x4 (0xF on pin reset) |
| ear | 0x0 |
| SYC | 0x0 |
| ECON | 0x0 |
| MIDX | 0x0 |
| MACH | 0xFF |
| MACL | 0xFFFF |

During periods of ECCP activity, write operations to the **eir** and **edr** registers as well as the read operation of the **edr** register by the DSP code will be blocked. The ECCP address register, **ear**, however, can be read or written during ECCP operation to set up the ECCP address for the next **edr** access after the completion of the ECCP instruction. Note that the **eir** register can be read during ECCP activity.

## 4 Hardware Architecture (continued)

**ECCP Internal Memory-Mapped Registers:** Internal memory-mapped registers are defined in the ECCP address space for control and status purposes and to hold data. A summary of the contents of these registers is given in Table 16.

**Table 16. Memory-Mapped Registers**

| Address | Register | Register Bit Field |
|---|---|---|
| 0x0000—0x007F | Next State Register<br>NS[0:63]—24-bit words split across two address locations | Bit 31:16 is addressed by even address.<br>Bit 31:24 is zero.<br>Bit 23:16 is the most significant byte of path cost.<br>Bit 15:0 is addressed by odd address.<br>Bit 15:0 is the lower 2 bytes of path cost. |
| 0x0080—0x01FF | Reserved | |
| 0x0200—0x027F | Present State Register<br>PS[0:63]—24-bit words split across two address locations | Bit 31:16 is addressed by even address.<br>Bit 31:24 is zero.<br>Bit 23:16 is the most significant byte of path cost.<br>Bit 15:0 is addressed by odd address.<br>Bit 15:0 is the lower 2 bytes of path cost. |
| 0x0280—0x03FF | Reserved | |
| 0x0400 | Current Symbol Pointer<br>SYC | Bit 5:0 is current symbol pointer.<br>Bit 15:6 reserved. |
| 0x0401 | Control Register<br>ECON | Bit 0 is soft-decision select.<br>Bit 1 is Manhattan/Euclidean branch metric select.<br>Bit 2 is soft/hard-decision select.<br>Bit 7:3 reserved.<br>Bit 10:8 is code rate select.<br>Bit 11 reserved.<br>Bit 14:12 is constraint length select.<br>Bit 15 reserved. |
| 0x0402 | Traceback Length Register<br>TBLR | Bit 5:0 is traceback length(0—63).<br>Bit 15:6 reserved. |
| 0x0403 | Received Symbol/Channel Tap Register<br>S5H5 | Convolutional decoding case:<br>Bit 7:0 reserved.<br>Bit 15:8 is S5.<br>MLSE equalization case:<br>Bit 7:0 is HQ5.<br>Bit 15:8 is HI5. |
| 0x0404 | Received Symbol/Channel Tap Register<br>S4H4 | Convolutional decoding case:<br>Bit 7:0 reserved.<br>Bit 15:8 is S4.<br>MLSE equalization case:<br>Bit 7:0 is HQ4.<br>Bit 15:8 is HI4. |
| 0x0405 | Received Symbol/Channel Tap Register<br>S3H3 | Convolutional decoding case:<br>Bit 7:0 reserved.<br>Bit 15:8 is S3.<br>MLSE equalization case:<br>Bit 7:0 is HQ3.<br>Bit 15:8 is HI3. |
| 0x0406 | Received Symbol/Channel Tap Register<br>S2H2 | Convolutional decoding case:<br>Bit 7:0 reserved.<br>Bit 15:8 is S2.<br>MLSE equalization case:<br>Bit 7:0 is HQ2.<br>Bit 15:8 is HI2. |

## 4 Hardware Architecture (continued)

**Table 16. Memory-Mapped Registers** (continued)

| Address | Register | Register Bit Field |
|---|---|---|
| 0x0407 | Received Symbol/Channel Tap Register<br>S1H1 | Convolutional decoding case:<br>Bit 7:0 reserved.<br>Bit 15:8 is S1.<br>MLSE equalization case:<br>Bit 7:0 is HQ1.<br>Bit 15:8 is HI1. |
| 0x0408 | Received Symbol/Channel Tap Register<br>S0H0 | Convolutional decoding case:<br>Bit 7:0 reserved.<br>Bit 15:8 is S0.<br>MLSE equalization case:<br>Bit 7:0 is HQ0.<br>Bit 15:8 is HI0. |
| 0x0409 | Decoded Symbol Register<br>DSR | Bit 7:0 is zero.<br>Bit 15:8 is decoded symbol. |
| 0x040A | Received Real Signal/Generating Polynomial<br>ZIG10 | Convolutional case:<br>Bit 7:0 is G0.<br>Bit 15:8 is G1.<br>MLSE case:<br>Bit 9:0 is in-phase part of received signal.<br>Bit 15:10 reserved. |
| 0x040B | Received Imaginary Signal/Generating Polynomial<br>ZQG32 | Convolutional case:<br>Bit 7:0 is G2.<br>Bit 15:8 is G3.<br>MLSE case:<br>Bit 9:0 is quadrature-phase part of received signal.<br>Bit 15:10 reserved. |
| 0x040C | Generating Polynomial<br>G54 | Convolutional case:<br>Bit 7:0 is G4.<br>Bit 15:8 is G5.<br>MLSE case:<br>Bit 15:0 reserved. |
| 0x040D | Minimum Cost Index Register<br>MIDX | Bit 7:0 is minimum state index.<br>Bit 15:8 reserved. |
| 0x040E—F | Minimum Accumulated Cost Register<br>MACH<br>MACL | 0x040E<br>Bit 15:8 is zero.<br>Bit 7:0 is upper byte of the minimum accumulated cost 0x040F.<br>Bit 15:0 is the lower 2 bytes of the minimum accumulated cost. |
| 0x0410 | Traceback Shift Register<br>TBSR | Traceback shift register (TBSR)<br>Bit 7:0 traceback decoded state left-aligned.<br>Bit 15:8 reserved. |
| 0x0411—0x07FF | Reserved | Reserved. |

## 4 Hardware Architecture (continued)

### 4.14 Control Register (ECON)

The constraint length, code rate, soft/hard-decision mode, branch metric select, and soft-decision data selection are set in the control register memory-mapped at address location 0x401. The bit allocation of the control register is the following.

**Table 17. Control Fields of the Control Register (ECON)**

| **Bits** | 15 | 14—12 | 11 | 10—8 | 7—3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Field** | Reserved | Constraint Length | Reserved | Code Rate | Reserved | SH | MAN | SD |

**Constraint Length**

The constraint length, L, sets the number of states in the Viterbi decoding process to $2^{L-1}$. The constraint length sets the number of bits in the generating polynomials for convolutional decoding and the number of complex channels estimate FIR taps for MLSE equalization. The constraint length also determines the effective length of the traceback shift register and the traceback RAM used to store the survivor paths.

Three bits in the control register set the constraint length for convolutional decoding or MLSE equalization. For hard-decision convolutional decoding, constraint lengths from 2 to 7 are supported. Hard-decision MLSE equalization is possible for constraint lengths from 2 to 6. For soft-decision convolutional decoding or MLSE equalization, constraint lengths from 2 to 6 are supported. This constraint length field is defined in the following table:

| **Bits ECON(14—12)** | **Constraint Length** | **# of PS/NS Registers** |
|---|---|---|
| 000 | 2 | 2 |
| 001 | 3 | 4 |
| 010 | 4 | 8 |
| 011 | 5 | 16 |
| 100 | 6 | 32 |
| 101 | 7 | 64 |
| 110 | Reserved | |
| 111 | Reserved | |

## 4 Hardware Architecture (continued)

### Code Rate

Three bits in the control register set the code rate of the convolutional decoder. The ECCP supports six different code rates for convolutional decoding. The code rate field is defined in the following table:

| Bits ECON(10—8) | Select Code Rate | Generating Polynomials | Symbols |
|---|---|---|---|
| 000 | 1/1 | G(0) | S(0) |
| 001 | 1/2 | G(0)—G(1) | S(0)—S(1) |
| 010 | 1/3 | G(0)—G(2) | S(0)—S(2) |
| 011 | 1/4 | G(0)—G(3) | S(0)—S(3) |
| 100 | 1/5 | G(0)—G(4) | S(0)—S(4) |
| 101 | 1/6 | G(0)—G(5) | S(0)—S(5) |
| 110 | Reserved | | |
| 111 | Reserved | | |

### Soft/Hard-Decision

The SH field of the control register sets the data packing mode in the traceback unit. The two options are to pack soft-decision data in a byte packed form, or hard-decision bits in a bit-packed mode.

| Bit ECON(2) SH | Function |
|---|---|
| 1 | Generate hard-decision bits as output |
| 0 | Generate 8-bit soft-decision as output |

### Rate 1/1 & 1/2 Metric Select

For convolutional decoding of rate 1/1 and 1/2, the branch metric is selected to be either the sum of squares or the Manhattan metric. The selection is set in bit 1 (MAN) of the ECON register.

| Bit ECON(1) MAN | Function |
|---|---|
| 0 | Select Euclidean Metric |
| 1 | Select Manhattan Metric |

### Soft Decode

Soft decode, SD, bit 0 of the control register selects one of two possible soft symbol definitions. The soft-decision data is set to the coded surviving branch metric, or to the coded absolute value of the difference between the surviving and rejected accumulated path cost.

| Bit ECON(0) SD | Function |
|---|---|
| 0 | Soft symbol is the coded accumulated cost difference, symbol is the traceback bit. |
| 1 | Soft symbol is the coded survivor branch metric, symbol is the MSB of the traceback shift register. |

### Soft-Decoded Output Definition

Two types of 8-bit soft-decoded output are implemented. One is the coded accumulated path cost difference, and the other is the coded survivor incremental branch metric.

**Coded Path Cost Difference:** An 8-bit quantized soft output is obtained from the accumulated cost difference of the two paths reaching a certain node in the trellis. The accumulated cost difference is a 24-bit binary number. Eight least significant bits of the absolute value of the difference, SD, are discarded. If the result is greater than 0x7F, it is saturated to 0x7F. The soft-decoded symbol, SS, is obtained from the hard-decision bit, TB, defined as the LSB of the present state, as follows:

$$SS = (0x7F - SD >> 8) \text{ if } TB = 0$$

else:

$$SS = 2^7 + SD >> 8 \text{ if } TB = 1$$

**Coded Survivor Branch Metric:** Another 8-bit quantized confidence measure of the soft-decoded output is obtained from the branch metric, BM, of the survivor transition. The 16-bit branch metric is scaled down with 9-bit right shift. If the decision bit, the most recent bit, is a 0, the soft-decoded output is:

$$SS = BM >> 9$$

else:

$$SS = 0xFF - BM >> 9$$

## 4 Hardware Architecture (continued)

**Current Symbol Register (SYC):** The physical pointer to the traceback memory is monitored and reported in the current symbol register at address location 0x0400. This is the address pointer used to address a particular symbol section in the traceback memory, which is shared with the thirtieth bank of the internal RAM, RAM30. This pointer is incremented after each UpdateMLSE and UpdateConv instruction. It is a modulo 32 count for soft symbol decision and modulo 64 count pointer for hard symbol count.

| **SYC Bits** | 15—6 | 5—0 |
|---|---|---|
| **Function** | Reserved | Current symbol pointer |

**Traceback Length Register (TBLR):** The traceback decoding length is stored in the traceback length register at address location 0x402. The traceback length is programmed by setting the TBLR field. When an UpdateMLSE or UpdateConv instruction is executed, a state update is processed. Also, a parallel traceback is processed from the last written symbol in the traceback memory addressed by minimum cost index register, going back through a number of symbols equal to the traceback length field. The user can change the traceback length from symbol to symbol. When a TraceBack instruction is executed, a simple traceback is processed starting at the state pointed to by the minimum cost index register, going back through a number of symbols equal to the traceback length field. The programmed traceback length field is automatically decremented by 1. TBLR should not be written with a value of 0. This results in incorrect traceback decoding operation. In addition, in the soft-decision mode, ECON.SH = 0, only values in the range of 1 to 31 are legal, while in the hard-decision mode, ECON.SH = 1, only values in the range of 1 to 63 are legal.

| **TBLR Bits** | 15—6 | 5—0 |
|---|---|---|
| **Function** | Reserved | Traceback length(1—63) |

**Minimum Cost State Index Register (MIDX):** The initial state number for traceback is stored in the minimum cost state index register at address location 0x040D. After an update instruction is completed, this register is automatically loaded with the state index corresponding to the minimum accumulated cost of the survivor paths determined in the update unit. Prior to a traceback or update instruction, the user can change the initial state index by writing to this register.

| **MIDX Bits** | 15—8 | 7—0 |
|---|---|---|
| **Function** | Reserved | Minimum state index |

**Traceback Shift Register (TBSR):** The shift register in the traceback unit used to address the traceback memory is located at address 0x0410. The number of significant bits is the constraint length minus one. The LSB of the traceback shift register is right aligned to bit 0. The register contains the latest L – 1 decoded bits.

| **TBSR Bits** | 15—8 | 7—0 |
|---|---|---|
| **Function** | Reserved | Traceback decoded state left-aligned |

**Update Cost Registers (NS[63:0], PS[63:0]):** Two blocks, each having 64 registers, are allocated for storing the accumulated path costs. Each register is 24 bits wide. Functionally, one block is the next state accumulated cost register bank, and the second block is the present state accumulated cost register bank. Next state registers NS[63:0] are located at 0x0000—0x007F, and present state registers PS[63:0] are located at 0x2000—0x027F. Two consecutive addresses are allocated to access each of these 24-bit registers. The even addresses starting with address zero access bits 23 to 16 of an update field padded with eight 0s at the upper byte, and the odd addresses access bits 15 to 0 of the same update field.

## 4 Hardware Architecture (continued)

**Generating Polynomial Registers (ZIG10, ZQG32, and G54):** For convolutional decoding, up to six generating polynomials are stored in three registers at address locations 0x040A to 0x040C. Odd-numbered generating polynomials are stored in the upper bytes of these three registers, and the even-numbered generating polynomials are stored in the lower 3 bytes of these registers.

Six generating polynomials support up to rate 1/6 convolutional decoding. The 6 bits of the generating polynomials (designated $D^1$ to $D^6$) support convolutional decoding up to a constraint length of 7. $D^1$, most recent delay, is aligned with the MSB of the appropriate generating polynomial registers. $D^0$ is assumed to always equal 1. Depending on the code rate set in the control register, the appropriate number of generating polynomials are used in the branch metric calculation.

| ZIG10 Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9—8 | 7 | 6 | 5 | 4 | 3 | 2 | 1—0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Function | G1 ($D^1$) | G1 ($D^2$) | G1 ($D^3$) | G1 ($D^4$) | G1 ($D^5$) | G1 ($D^6$) | Reserved | G0 ($D^1$) | G0 ($D^2$) | G0 ($D^3$) | G0 ($D^4$) | G0 ($D^5$) | G0 ($D^6$) | Reserved |

| ZQG32 Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9—8 | 7 | 6 | 5 | 4 | 3 | 2 | 1—0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Function | G3 ($D^1$) | G3 ($D^2$) | G3 ($D^3$) | G3 ($D^4$) | G3 ($D^5$) | G3 ($D^6$) | Reserved | G2 ($D^1$) | G2 ($D^2$) | G2 ($D^3$) | G2 ($D^4$) | G2 ($D^5$) | G2 ($D^6$) | Reserved |

| G54 Bits | 15 | 14 | 13 | 12 | 11 | 10 | 9—8 | 7 | 6 | 5 | 4 | 3 | 2 | 1—0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Function | G5 ($D^1$) | G5 ($D^2$) | G5 ($D^3$) | G5 ($D^4$) | G5 ($D^5$) | G5 ($D^6$) | Reserved | G4 ($D^1$) | G4 ($D^2$) | G4 ($D^3$) | G4 ($D^4$) | G4 ($D^5$) | G4 ($D^6$) | Reserved |

**Decoded Symbol Register (DSR):** The decoded symbol register at address location 0x0409 stores the symbol generated by the ECCP traceback unit. At the end of a TraceBack, an UpdateConv, or UpdateMLSE instruction, a decoded symbol is generated and saved in the upper byte of the decoded symbol register. In hard-decoded symbol mode, bit 15 represents the decoded symbol, and, in soft-decoded symbol mode, bits 15—8 represent the soft symbol.

| DSR Bits | 15—8 | 7—0 |
|---|---|---|
| Function | Soft Decoded Symbol | 0 |

| DSR Bits | 15 | 14—0 |
|---|---|---|
| Function | Hard Decoded Symbol | 0 |

**Binary Magnitude Symbol/Channel Model Registers ($S_iH_i$, i = 0, 1, . . . , 5):** The symbol registers consist of six words at address locations 0x0403 to 0x0408, the contents of which are used for branch metric calculations. For convolutional decoding, the upper bytes of these six words contain received symbols, $S_i(n)$, **i** = 0, 1, . . . , 5, in 8-bit binary magnitude form. For MLSE equalization, the high byte stores in-phase channel estimate coefficients HI(n) in 8-bit 2's complement form, and the low byte stores the quadrature components HQ(n) in 8-bit 2's complement form.

| $S_iH_i$ Bits | 15—8 | 7—0 |
|---|---|---|
| MLSE Function | $HI_i$ | $HQ_i$ |
| Conv Function | $S_i$ | Reserved |

## 4 Hardware Architecture (continued)

**Complex Received Symbol Registers (ZIG10, ZQG32):** The complex received symbol registers are used for MLSE equalization. The complex received symbol is stored in two registers, each in 10-bit 2's complement form. The in-phase part of the received symbol is stored in the lower 10 bits of address location 0x040A and the quadrature-phase part of the received symbol is stored in the lower 10 bits of address location 0x040B.

| **ZIG10 Bits** | 15—10 | 9—0 |
|---|---|---|
| **Function** | Reserved | ZI |

| **ZQG32 Bits** | 15—10 | 9—0 |
|---|---|---|
| **Function** | Reserved | ZQ |

**Reserved Registers:** Addresses above 0x0410 are reserved and should not be accessed by the user code. Specifically, a write to **edr** with **ear** containing addresses higher than 0x0410 can result in the incorrect operation of the ECCP.

**ECCP Interrupts and Flags:** The ECCP interrupts the DSP core with two vectored interrupts and its status is indicated with a user flag.

The ECCP user flag is EBUSY. This flag is used in conjunction with the **if CON F2** or **if CON goto/call/return** instructions to monitor the ECCP status during ECCP operation. The flag is defined as:

- **EBUSY:** Asserted when the **eir** is written with an UpdateMLSE, UpdateConv, or TraceBack instruction. Negated when the ECCP instruction is completed. When EBUSY flag is asserted, read operations of the **edr** register and write operations to the **eir** and **edr** registers, including **eir = ResetECCP**, are ignored. Also, RAM30 cannot be accessed.

Two vectored interrupts are EREADY and EOVF. These interrupts are maskable through the **inc** register and their status can be read or changed using the **ins** register using the DSP1600 interrupt conventions. An **ireturn** from the vectored interrupt service routine clears the interrupt status. See Section 4.3, Interrupts and Trap, for further discussion. The interrupts are defined as follows:

- **EREADY:** Asserted three cycles before the EBUSY flag is negated. Negated upon writing a 1 in the EREADY field of the **ins** register, or upon executing an **ireturn**.
- **EOVF:** An overflow condition is detected when any one of the next state registers is loaded with 0xFF in the eight MSBs. This EOVF interrupt is then asserted to the DSP only after the current instruction is completed. EOVF is negated upon writing a 1 in the EOVF field of the **ins** register, or upon executing an **ireturn**.

**Traceback RAM:** The thirtieth 1 Kword bank of dual-port RAM is shared by the ECCP for storing the traceback information. When the ECCP is active, i.e., the EBUSY flag is asserted, the DSP core cannot access this traceback RAM. DSP write operations to RAM30 are ignored and read operations access corrupted data. As a rule, if the **eir** register is written with one of the UpdateMLSE, UpdateConv, or TraceBack instructions, the DSP software must avoid accessing RAM30 from either the X-side or Y-side. Following one of these instructions, the software can determine the end of ECCP activity either by polling the EBUSY flag and waiting for its negation or by waiting for the EREADY interrupt to be asserted. In the later case, RAM30 can be accessed by the EREADY interrupt service routine.

## 4 Hardware Architecture (continued)

**Programming Limitations:** Although in general it is not recommended, user data as well as user code can reside in RAM30. Also, the user code that programs the ECCP and writes the instruction register, **eir**, can be executed from RAM30. However, several programming restrictions are imposed on such blocks of code and data:

1. The location of the user code must not conflict with the addresses in RAM30 used for the storage of traceback information. The ECCP uses RAM30's address range 0x7400 to 0x7400 + $2^{(CL + 5)}$ in the soft-decision mode (i.e., when ECON.SH = 0) and the address range 0x7400 to 0x7400 + $2^{(CL + 3)}$ in the hard-decision mode (i.e., when ECON.SH = 1) for the storage of traceback data, where CL represents the value of the constraint length field of the ECON register. Any user data or code in RAM30 must reside outside of these address ranges.

2. Access to RAM30 data and execution of code from RAM30 can be performed only during periods of ECCP inactivity. The only exception to this rule is the execution of ECCP instructions from RAM30. A jump to memory locations outside RAM30's address range must occur immediately after the loading of the **eir** instruction register as illustrated by the following code fragment:

```
.rsect ".ram"           /* ECCP code to reside in RAM */

OutofRAM30:             /* This address is outside of RAM30 */
if ebusy goto .         /* Wait for ECCP to finish */
...                     /* Now can access ECCP and/or RAM30 */


.=0x7400 + offset       /* Offset to avoid conflict with ECCP */
program_eccp:
...                     /* Load various ECCP registers here */
eir = UpdateMLSE        /* Invoke ECCP instruction */
pt = OutofRAM30         /* Address outside RAM30 */
goto pt                 /* Jump out of RAM30 */
```

### ECCP Instruction Timing

**ECCP Data Move Timing:** Each ECCP data move instruction takes two cycles.

**Viterbi Instruction Timing:** Six different categories of instructions are presented here for the ease of the instruction cycle formulation.

**ResetECCP Instruction:** The ResetECCP instruction has no latency.

**UpdateMLSE Instruction with Soft-Decision:** The generic formula for the computation of the UpdateMLSE instruction cycles with soft-decision (i.e., SH = 0) is as follows:

$$\text{UpdateMLSE (SH = 0) Cycles} = 15 + 2^{(CL + 2)} + \text{Max}[0, \text{TBLR} - 2^{(CL + 2)} + 2^{CL} - 4]$$

where CL represents the value of the constraint length field in the ECON register and TBLR is the traceback length value programmed into the TBLR register.

Table 18 shows some representative values for the UpdateMLSE instruction cycles for different values of CL and TBLR. Note that for the UpdateMLSE instruction, CL has a maximum value of four, corresponding to constraint length 6.

Note that for the UpdateMLSE instruction with soft-decision, the traceback length register can be programmed to a maximum value of 31. TBLR values greater than 31 are illegal and must not be used along with the Update MLSE instruction when soft-decision mode is selected.

## 4 Hardware Architecture (continued)

**Table 18. Representative UpdateMLSE Instruction Cycles (SH = 0)**

| CL | TBLR | Cycles |
|---|---|---|
| 0 | 1—7 | 19 |
| 0 | 8 | 20 |
| 0 | 9 | 21 |
| 0 | 10 | 22 |
| 0 | 11 | 23 |
| 1 | 1—10 | 23 |
| 1 | 11 | 24 |
| 1 | 12 | 25 |
| 1 | 13 | 26 |
| 1 | 14 | 27 |
| 2 | 1—16 | 31 |
| 2 | 17 | 32 |
| 2 | 18 | 33 |
| 2 | 19 | 34 |
| 2 | 20 | 35 |
| 3 | 1—28 | 47 |
| 3 | 29 | 48 |
| 3 | 30 | 49 |
| 3 | 31 | 50 |
| 4 | 1—31 | 79 |

**UpdateMLSE Instruction with Hard-Decision:**

The generic formula for the computation of the UpdateMLSE instruction cycles with hard-decision (i.e., SH = 1) is as follows:

$$\text{UpdateMLSE(SH = 1) Cycles} = 15 + 2^{(CL + 2)} + \text{Max}[0, (\text{TBLR} - 2^{(CL + 2)} + \text{Max}[1, 2^{(CL - 3)}] - 4)]$$

where CL represents the value of the constraint length field in the ECON register and TBLR is the traceback length value programmed into the TBLR register. Table 19 shows some representative values for the UpdateMLSE instruction cycles for different values of CL and TBLR. Note that for the UpdateMLSE instruction, CL has a maximum value of four corresponding to constraint length 6.

**Table 19. Representative UpdateMLSE Instruction Cycles (SH = 1)**

| CL | TBLR | Cycles |
|---|---|---|
| 0 | 1—7 | 19 |
| 0 | 8 | 20 |
| 0 | 9 | 21 |
| 0 | 10 | 22 |
| 0 | 11 | 23 |
| 1 | 1—11 | 23 |
| 1 | 12 | 24 |
| 1 | 13 | 25 |
| 1 | 14 | 26 |
| 1 | 15 | 27 |
| 2 | 1—19 | 31 |
| 2 | 20 | 32 |
| 2 | 21 | 33 |
| 2 | 22 | 34 |
| 2 | 23 | 35 |
| 3 | 1—35 | 47 |
| 3 | 36 | 48 |
| 3 | 37 | 49 |
| 3 | 38 | 50 |
| 4 | 1—63 | 79 |

Note that for the UpdateMLSE instruction with hard-decision, the traceback length register can be programmed to a maximum value of 63.

**UpdateConv Instruction with Soft-Decision:** With the ECON.SH field set to 0, i.e., with soft-decision mode selected, the following formula yields the number of instruction cycles for the UpdateConv instruction:

$$\text{UpdateConv(SH = 0) Cycles} = 14 + 2^{(CL + 2)} + \text{Max}[0, \text{TBLR} - 2^{(CL + 2)} + 2^{CL} - 3]$$

where CL represents the value of the constraint length field in the ECON register, and TBLR is the traceback length value programmed into the TBLR register. The following table shows some representative values for the UpdateConv instruction cycles with the soft-decision mode selected for different values of CL and TBLR.

## 4 Hardware Architecture (continued)

**Table 20. Representative UpdateConv Instruction Cycles (SH = 0)**

| CL | TBLR | Cycles |
|---|---|---|
| 0 | 1—6 | 18 |
| 0 | 7 | 19 |
| 0 | 8 | 20 |
| 0 | 9 | 21 |
| 0 | 10 | 22 |
| 1 | 1—9 | 22 |
| 1 | 10 | 23 |
| 1 | 11 | 24 |
| 1 | 12 | 25 |
| 1 | 13 | 26 |
| 2 | 1—15 | 30 |
| 2 | 16 | 31 |
| 2 | 17 | 32 |
| 2 | 18 | 33 |
| 2 | 19 | 34 |
| 3 | 1—27 | 46 |
| 3 | 28 | 47 |
| 3 | 29 | 48 |
| 3 | 30 | 49 |
| 3 | 31 | 50 |
| 4 | 1—31 | 78 |

Note that, similar to the UpdateMLSE, the traceback length can attain a maximum value of 31 with the soft-decision mode programmed (i.e., with ECON.SH = 0).

**UpdateConv Instruction with Hard-Decision:** With the ECON.SH field set to 1, i.e., with hard-decision mode selected, the following formula yields the number of instruction cycles for the UpdateConv instruction.

$$\text{UpdateConv(SH = 1) Cycles} = 14 + 2^{(CL + 2)} + \text{Max}[0, (\text{TBLR} - 2^{(CL + 2)} + \text{Max}[1, 2^{(CL - 3)}] - 3)]$$

where CL represents the value of the constraint length field in the ECON register and TBLR is the traceback length value programmed into the TBLR register. The following table shows some representative values for the UpdateConv instruction cycles with hard-decision mode selected for different values of CL and TBLR.

**Table 21. Representative UpdateConv Instruction Cycles (SH = 1)**

| CL | TBLR | Cycles |
|---|---|---|
| 0 | 1—6 | 18 |
| 0 | 7 | 19 |
| 0 | 8 | 20 |
| 0 | 9 | 21 |
| 0 | 10 | 22 |
| 1 | 1—10 | 22 |
| 1 | 11 | 23 |
| 1 | 12 | 24 |
| 1 | 13 | 25 |
| 1 | 14 | 26 |
| 2 | 1—18 | 30 |
| 2 | 19 | 31 |
| 2 | 20 | 32 |
| 2 | 21 | 33 |
| 2 | 22 | 34 |
| 3 | 1—34 | 46 |
| 3 | 35 | 47 |
| 3 | 36 | 48 |
| 3 | 37 | 49 |
| 3 | 38 | 50 |
| 4 | 1—63 | 78 |
| 5 | 1—63 | 142 |

Note that the traceback length register can reach a maximum value of 63 with the hard-decision decoding mode selected.

**TraceBack Instruction:** The length of the TraceBack instruction is only a function of the programmed trace-back length and is equal to:

$$\text{Traceback Cycles} = \text{TBLR} + 14.$$

Note that the TBLR can be programmed to a maximum value of 31, if the TraceBack instruction is used after UpdateMLSE instructions or after UpdateConv instructions with soft-decision symbols. A maximum value of 63 can be programmed for hard-decision decoding after UpdateMLSE or UpdateConv instructions. Also, the contents of the TBLR register are autodecremented after the TraceBack instruction is completed.

## 4 Hardware Architecture (continued)

### 4.15 JTAG Test Port

The DSP1620 uses a JTAG/*IEEE* 1149.1 standard five-wire test port (TDI, TDO, TCK, TMS, TRST) for self-test and hardware emulation. An instruction register, a boundary-scan register, a bypass register, and a device identification register have been implemented. The device identification register coding for the DSP1620 is shown in Table 38. The instruction register (IR) is 4 bits long. The instruction for accessing the device ID is 0xE (1110). The behavior of the instruction register is summarized in Table 22. Cell 0 is the LSB (closest to TDO).

**Table 22. JTAG Instruction Register**

| IR Cell #: | 3 | 2 | 1 | 0 |
|---|---|---|---|---|
| parallel input? | Y | Y | N | N |
| always logic 1? | N | N | N | Y |
| always logic 0? | N | N | Y | N |

The first line shows the cells in the IR that capture from a parallel input in the capture-IR controller state. The second line shows the cells that always load a logic 1 in the capture-IR controller state. The third line shows the cells that always load a logic 0 in the capture-IR controller state. Cell 3 (MSB of IR) is tied to status signal PINT, and cell 2 is tied to status signal JINT. The state of these signals can therefore be captured during capture-IR and shifted out during SHIFT-IR controller states.

#### Boundary-Scan Register

All of the chip's inputs and outputs are incorporated in a JTAG scan path shown in Table 23. The types of boundary-scan cells are as follows:

- I = input cell
- O = 3-state output cell
- B = bidirectional (I/O) cell
- OE = 3-state control cell
- DC = bidirectional control cell

## 4 Hardware Architecture (continued)

**Table 23. JTAG Boundary-Scan Register**

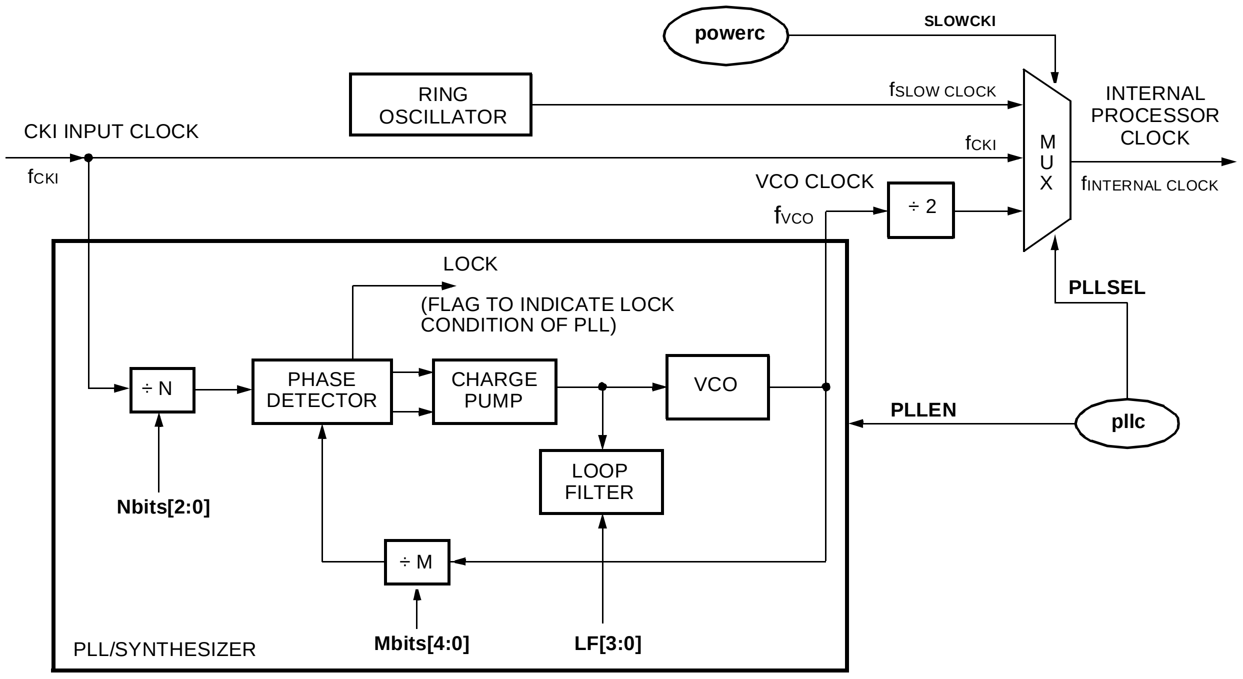
**Note:** The direction of shifting is from TDI to cell 127 to cell 126 . . . to cell 0 to TDO.

| Cell | Type | Signal Name/Function | Cell | Type | Signal Name/Function |
|---|---|---|---|---|---|
| 0 | I | CKI* | 78 | DC | PB[15:8] |
| 1 | OE | AB, IACK, PIBF, POBE, OBE1, IBF1, OBE2, IBF2 | 79—86 | B | PB[15:8] |
| 2—17 | O | AB[0:15] | 87 | O | OBE2 |
| 18 | I | EXM | 88 | O | IBF2 |
| 19 | OE | RWN, EROM, ERAMLO, ERAMHI, ERAMX, IO, CKO | 89 | I | DI2 |
| 20 | O | RWN | 90 | DC | ILD2 |
| 21 | O | EROM | 91 | B | ILD2 |
| 22 | O | ERAMLO | 92 | DC | ICK2 |
| 23 | O | ERAMHI | 93 | B | ICK2 |
| 24 | O | ERAMX | 94 | DC | OCK2 |
| 25 | O | IO | 95 | B | OCK2 |
| 26 | DC | DB | 96 | DC | OLD2 |
| 27—42 | B | DB[0:15] | 97 | B | OLD2 |
| 43 | O | OBE1 | 98 | DC | DO2 |
| 44 | O | IBF1 | 99 | O | DO2 |
| 45 | I | DI1 | 100 | I | DOEN2 |
| 46 | DC | ILD1 | 101 | DC | IOBIT0 |
| 47 | B | ILD1 | 102 | B | IOBIT0 |
| 48 | DC | ICK1 | 103 | DC | IOBIT1 |
| 49 | B | ICK1 | 104 | B | IOBIT1 |
| 50 | DC | OCK1 | 105 | DC | IOBIT2 |
| 51 | B | OCK1 | 106 | B | IOBIT2 |
| 52 | DC | OLD1 | 107 | DC | IOBIT3 |
| 53 | B | OLD1 | 108 | B | IOBIT3 |
| 54 | DC | DO1 | 109 | DC | VEC3/IOBIT4† |
| 55 | O | DO1 | 110 | B | VEC3/IOBIT4† |
| 56 | DC | SYNC1 | 111 | DC | VEC2/IOBIT5† |
| 57 | B | SYNC1 | 112 | B | VEC2/IOBIT5† |
| 58 | DC | SADD1 | 113 | DC | VEC1/IOBIT6† |
| 59 | B | SADD1 | 114 | B | VEC1/IOBIT6† |
| 60 | DC | DOEN1 | 115 | DC | VEC0/IOBIT7† |
| 61 | B | DOEN1 | 116 | B | VEC0/IOBIT7† |
| 62 | I | PIDS | 117 | I | READY |
| 63 | I | PCSN | 118 | I | STOP‡ |
| 64 | I | PSTAT | 119 | I | RSTB |
| 65 | I | PBSEL | 120 | O | CKO |
| 66 | I | PODS | 121 | DC | TRAP |
| 67 | O | PIBF | 122 | B | TRAP |
| 68 | O | POBE | 123 | O | IACK |
| 69 | DC | PB[7:0] | 124—127 | I | INT[3:0] |
| 70—77 | B | PB[7:0] | — | — | — |

* When the JTAG SAMPLE instruction is used, this cell will have a logic one regardless of the state of the pin.
† Refer to Pin Multiplexing in Section 4.1 for a description of pin multiplexing of IOBIT[4:7] and VEC[3:0].
‡ Note that shifting a zero into this cell in the mode to scan a zero into the chip will disable the processor clocks just as the STOP pin will.

## 4 Hardware Architecture (continued)

### 4.16 Clock Synthesis



5-4520(F)

Notes:
Signals shown in bold are control bits from the **pllc** register or the **powerc** register.

When PLLSEL = 0, DSP runs from the 1X version of CKI input clock.

Other signals from the **powerc** register also control the clock source.

**Figure 13. Clock Source Block Diagram**

The DSP1620 provides an on-chip, programmable clock synthesizer. Figure 13 is the clock source diagram. The 1X CKI input clock, the output of the synthesizer, or a slow internal ring oscillator can be used as the source for the internal DSP clock. The clock synthesizer is based on a phase-locked loop (PLL), and the terms clock synthesizer and PLL are used interchangeably.

On powerup, CKI is used as the clock source for the DSP. This clock is used to generate the internal processor clocks and CKO, where $f_{CKI} = f_{CKO}$. Setting the appropriate bits in the **pllc** control register (described in Table 44) enables the clock synthesizer to become the clock source. The **powerc** register, discussed in Section 4.17, can override the selection to stop clocks or force the use of the slow clock for low-power operation.

## 4 Hardware Architecture (continued)

### PLL Control Signals

The input to the PLL comes from the CKI input pin. The PLL cannot operate without an external input clock.

To use the PLL, the PLL must first be allowed to stabilize and lock to the programmed frequency. After the PLL has locked, the LOCK flag is set and the lock detect circuitry is disabled. The synthesizer can then be used as the clock source. Setting the PLLSEL bit in the **pllc** register switches sources from $f_{CKI}$ to $f_{VCO}/2$ without glitching. It is important to note that the setting of the **pllc** register must be maintained. Otherwise, the PLL seeks the new set point. The **pllc** register cannot be reconfigured while it is enabled.

The frequency of the PLL output clock, $f_{VCO}$, is determined by the values loaded into the 3-bit N divider and the 5-bit M divider. When the PLL is selected and locked, the frequency of the internal processor clock is related to the frequency of CKI by the following equations:

$$f_{VCO} = f_{CKI} * M/N$$

$$f_{\text{internal clock}} = f_{CKO} = f_{VCO} \div 2$$

The frequency of the VCO, $f_{VCO}$, must be at least twice $f_{CKI}$.

The coding of the Mbits and Nbits is described as follows:

```
Mbits = M – 2
if (N == 1)
    Nbits = 0x7
else
    Nbits = N – 2
```

where N ranges from 1 to 8 and M ranges from 2 to 24.

Two other bits in the **pllc** register control the PLL. Clearing the PLLEN bit powers down the PLL; setting this bit powers up the PLL. Clearing the PLLSEL bit deselects the PLL so that the DSP is clocked by a 1X version of the CKI input; setting the PLLSEL bit selects the PLL-generated clock for the source of the DSP internal processor clock. The **pllc** register is cleared on reset and powerup. Therefore, the DSP comes out of reset with the PLL deselected and powered down. M and N should be changed only while the PLL is deselected. The process for changing the values of M and N is as follows:

1. Deselect PLL
2. Change M and N and wait for lock (poll the LOCK flag)
3. Select PLL

As previously mentioned, the PLL also provides a user flag, LOCK, to indicate when the loop has locked. When this flag is not asserted, the PLL output is unstable. The DSP should not be switched to the PLL-based clock without first checking that the LOCK flag is set. The LOCK flag is cleared by writing to the **pllc** register. When the PLL is deselected, it is necessary to wait for the PLL to relock before the DSP can be switched to the PLL-based clock. Before the input clock is stopped, the PLL should be powered down. Otherwise, the LOCK flag is not reset and there is no way to determine if the PLL is stable, once the input clock is applied again.

The lock-in time depends on the frequency of operation and the values programmed for M and N.

## 4 Hardware Architecture (continued)

### PLL Programming Examples

The following section of code illustrates how the PLL would be initialized on powerup, assuming the following operating conditions:

- CKI input frequency = .......... 10 MHz
- Internal clock and CKO frequency = .......... 50 MHz
- VCO frequency = .......... 100 MHz
- Input divide down count N = .......... 2 (Set **Nbits[2:0]** = 000 to get N = 2, as described in Table 44.)
- Feedback down count M = ........ 20 (Set **Mbits[4:0]** = 10010 to get M = 18 + 2 = 20, as described in Table 44.)

The device would come out of reset with the PLL disabled and deselected.

```
pllinit:pllc = 0x2912/* Running CKI input clock at 10 MHz, set up counters in PLL */
      pllc = 0xA912 /*  Power on PLL, but PLL remains deselected */
      call pllwait/*    Loop to check for LOCK flag assertion */
      pllc = 0xE912 /*  Select high-speed, PLL clock */
      goto start  /*    User's code, now running at 50 MHz */
pllwait:if lock return
      goto pllwait
```

Programming examples that illustrate how to use the PLL with the various power management modes are listed in Section 4.17.

### Latency

The switch between the CKI-based clock and the PLL-based clock is synchronous. This method results in the actual switch taking place several cycles after the PLLSEL bit is changed. During this time, actual code can be executed, but it is at the previous clock rate. Table 24 shows the latency times for switching between CKI-based and PLL-based clocks. In the example given, the delay to switch to the PLL source is 1—4 CKO cycles and to switch back is 11—31 CKO cycles.

**Table 24. Latency Times for Switching Between CKI- and PLL-Based Clocks**

| | Minimum Latency (Cycles) | Maximum Latency (Cycles) |
|---|---|---|
| **Switch to PLL-Based Clock** | 1 | N + 2 |
| **Switch from PLL-Based Clock** | M/N + 1 | M + M/N + 1 |

### Frequency Accuracy and Jitter

When using the PLL to multiply the input clock frequency up to the instruction clock rate, it is important to realize that although the average frequency of the internal clock and CKO has about the same relative accuracy as the input clock, noise sources within the DSP produce jitter on the PLL clock; therefore, each individual clock period has some error associated with it. The PLL is guaranteed to have only sufficiently low jitter to operate the DSP; therefore, this clock should not be used as an input to jitter-sensitive devices in the system.

### VDDA and VSSA Connections

The PLL has its own power and ground pins, VDDA and VSSA. Additional filtering should be provided for VDDA in the form of a ferrite bead connected from VDDA to VDD and two decoupling capacitors (4.7 µF tantalum in parallel with a 0.01 µF ceramic) from VDDA to VSS. VSSA can be connected directly to the main ground plane. This recommendation is subject to change and could be modified for specific applications depending on the characteristics of the supply noise.

# 4 Hardware Architecture (continued)

## 4.17 Power Management

There are three different control mechanisms for putting the DSP1620 into low-power modes: the **powerc** control register, the STOP pin, and the AWAIT bit in the **alf** register. The PLL can also be disabled with the PLLEN bit of the **pllc** register for more power saving.

### Powerc Control Register Bits

The **powerc** register has 9 bits that power down various portions of the chip and select the clock source:

SLOWCKI: If the program sets the SLOWCKI bit, an internal ring oscillator is selected as the source for CLK instead of the CKI pin or the PLL. If the SLOWCKI 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 SLOWCKI.

NOCK: If the program sets the NOCK bit, the DSP1620 synchronously turns off the internal processor clock (regardless of whether its source is provided by the CKI pin, the PLL, or the internal ring oscillator) and halts 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), allowing the halted program to resume execution from where it left off without any loss of state. If INT0EN or INT1EN is set, to avoid an unintentional interrupt due to the subsequent assertion of the INT0 or INT1 pin, it is recommended that the programmer disable the corresponding interrupt in the **inc** register before setting NOCK. After the halted program resumes, it should clear the corresponding INT0/INT1 interrupt by writing to the **ins** register (see Clearing Interrupts on page 24). Resetting the DSP1620 by asserting the RSTB pin also clears the NOCK bit, but the halted program cannot resume execution.

**Note:** If the PLL is enabled, it remains running while NOCK is set. For maximum power savings it is recommended that the programmer clear PLLEN prior to setting NOCK.

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 power down the peripheral I/O units of the DSP. These bits can be used to further reduce the power consumption during standard sleep mode.

SIO1DIS: This is a powerdown signal to the SIO I/O unit. It disables the clock input to the unit, thus eliminating any sleep power associated with the SIO. Since the gating of the clocks can result in incomplete transactions, it is recommended that this option be used in applications where the SIO is not used or when reset is used to reenable the SIO unit. Otherwise, the first transaction after reenabling the unit could be corrupted.

SSIODIS: This bit powers down the SSIO and MIOU0 in the same way SIO1DIS powers down the SIO.

PHIFDIS: This is a powerdown signal to PHIF16 and MIOU1. It disables the clock input to the unit, thus eliminating any sleep power associated with the PHIF16. Since the gating of the clocks can result in incomplete transactions, it is recommended that this option be used in applications where the PHIF16 is not used, or when reset is used to reenable the PHIF16.

TIMERDIS: This is a timer disable signal that disables the clock input to the timer unit. Its function is identical to the DISABLE field of the **timerc** control register. Writing a 0 to the TIMERDIS field will continue the timer operation.

ECCPDIS: This bit powers down the ECCP. It disables the clock input to the ECCP, thus eliminating any sleep power associated with the coprocessor. This bit cannot be used in applications where the ECCP is used.

Figure 14 shows a functional view of the effect of the bits of the **powerc** register on the clock circuitry. It shows only the high-level operation of each bit. Not shown are the bits that power down the peripheral units.

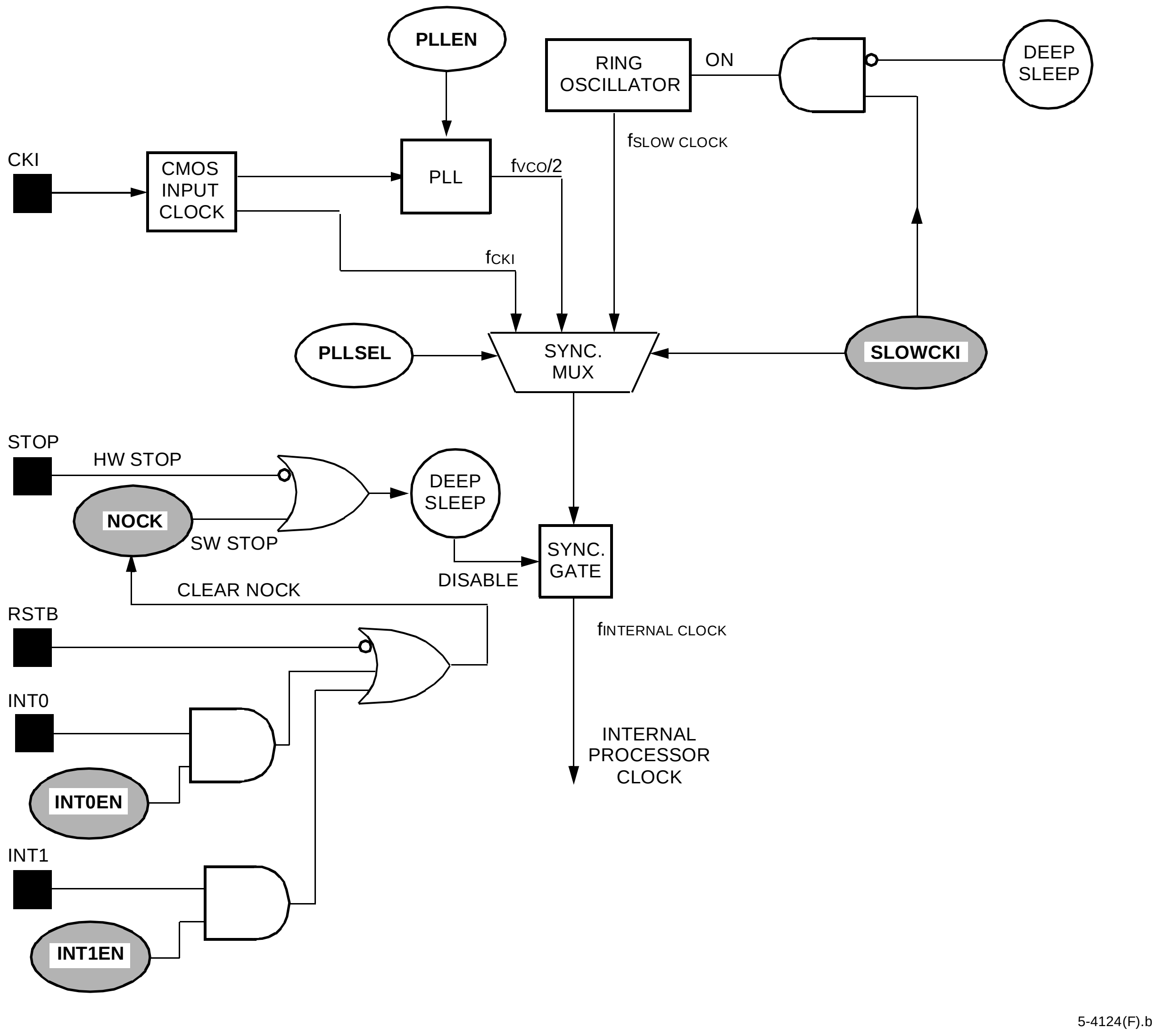
### STOP Pin

Assertion (active-low) of the STOP pin has the same effect as setting the NOCK bit in the **powerc** register. The internal processor clock is synchronously disabled until the STOP pin is returned high. Once the STOP pin is returned high, program execution will continue from where it left off without any loss of state. No chip reset is required. The PLL remains running, if enabled, during STOP assertion.

### The pllc Register Bits

The PLLEN bit of the **pllc** register can be used to power down the clock synthesizer circuitry. Before shutting down the clock synthesizer circuitry, the system clock should be switched to either CKI using the PLLSEL bit of **pllc**, or to the ring oscillator using the SLOWCKI bit of **powerc**.

## 4 Hardware Architecture (continued)



Notes:
The functions in the shaded ovals are bits in the **powerc** control register. The functions in the nonshaded ovals are bits in the **pllc** control register.

Deep sleep is the state arrived at either by a hardware or software stop of the internal processor clock.

The switching of the multiplexers and the synchronous gate is designed so that no partial clocks or glitching will occur.

When the deep sleep state is entered with the ring oscillator selected, the internal processor clock is turned off before the ring oscillator is powered down.

PLL select is the PLLSEL bit of **pllc**; PLL powerdown is the PLLEN bit of **pllc**.

**Figure 14. Power Management Using the powerc and the pllc Registers**

## 4 Hardware Architecture (continued)

### AWAIT Bit of the alf Register

Setting the AWAIT bit of the **alf** register causes the processor to go into the standard sleep state or power-saving standby mode. Operation of the AWAIT bit is the same as in the DSP1627. In this mode, the minimum circuitry required to process an incoming interrupt remains active, and the PLL remains active if enabled. An interrupt will return the processor to the previous state, and program execution will continue. The action resulting from setting the AWAIT bit and the action resulting from setting bits in the **powerc** register are mostly independent. As long as the processor is receiving a clock, whether slow or fast, the DSP can be put into standard sleep mode with the AWAIT bit. Once the AWAIT bit is set, the STOP pin can be used to stop and later restart the processor clock, returning to the standard sleep state. If the processor clock is not running, however, the AWAIT bit cannot be set.

### Power Management Sequencing

There are important considerations for sequencing the power management modes. The PLL requires a delay to reach lock-in. Also, the chip might or might not need to be reset following a return from a low-power state.

### Power Management Examples Without the PLL

The following examples show the more significant options for reducing the power dissipation. These are valid only if the **pllc** register is set to disable and deselect the PLL (PLLEN = 0, PLLSEL = 0).

**Standard Sleep Mode.** This is the standard sleep mode. While the processor is clocked with a high-speed clock, CKI, the **alf** register's AWAIT bit is set. Peripheral units can be turned off to further reduce the sleep power.

```
      powerc = 0x00F0   /* Turn off peripherals, core running with CKI */
sleep:a0 = 0x8000       /* Set alf register in cache loop if running from */
      do 1 {            /* external memory with >1 wait-state */
      alf = a0          /* Stop internal processor clock, interrupt circuits */
      nop               /* active  */
      }
      nop               /* Needed for bedtime execution. Only sleep power */
      nop               /* consumed here until.... interrupt wakes up the device */
cont: .  .  .           /* User code executes here */
      powerc = 0x0      /* Turn peripheral units back on */
```

**Sleep with Slow Internal Clock.** In this case, the ring oscillator is selected to clock the processor before the device is put to sleep. This reduces the power dissipation while waiting for an interrupt to continue program execution.

```
      powerc = 0x40F0   /* Turn off peripherals and select slow clock */
      2*nop             /* Wait for it to take effect */
sleep:a0 = 0x8000       /* Set alf register in cache loop if running from */
      do 1 {            /* external memory with >1 wait-state */
      alf = a0          /* Stop internal processor clock, interrupt circuits */
      nop               /* active  */
      }
      nop               /* Needed for bedtime execution. Reduced sleep power */
      nop               /* consumed here.... Interrupt wakes up the device */
cont: .  .  .           /* User code executes here */
      powerc = 0x00F0   /* Select high-speed clock */
      2*nop             /* Wait for it to take effect */
      powerc = 0x0000   /* Turn peripheral units back on */
```

Note that, in this case, the wake-up latency is determined by the period of the ring oscillator clock.

## 4 Hardware Architecture (continued)

**Software Stop.** In this case, all internal clocking is disabled. INT0, INT1, or RSTB can be used to reenable the clocks.

```
      powerc = 0x4000   /* SLOWCKI asserted */
      2*nop             /* Wait for it to take effect */
      powerc = 0x5000   /* INT0EN asserted */
      inc = NOINT0      /* Disable the INT0 interrupt */
sopor:powerc = 0x7000   /* NOCK asserted, all clocks stop */
                        /* Minimum switching power consumed here */
      3*nop             /* Some nops will be needed */
                        /* INT0 pin clears the NOCK field, clocking resumes */
cont: powerc = 0x4000   /* INT0EN cleared*/
      2*nop             /* Wait for it to take effect*/
      powerc = 0x0      /* Clear SLOWCKI field, back to high speed */
      2*nop             /* Wait for it to take effect */
      ins = 0x0010      /* Clear the INT0 status bit */
```

The previous examples do not provide an exhaustive list of options available to the user. Many different clocking possibilities exist for which the target device can be programmed, depending on:

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

### Power Management Examples with the PLL

The following examples show the more significant options for reducing power dissipation if operation with the PLL clock synthesizer is desired.

**Standard Sleep Mode, PLL Running.** This mode would be entered in the same manner as without the PLL. While the input to the clock synthesizer, CKI, remains running, the **alf** register's AWAIT bit is set. The PLL will continue to run and dissipate power. Peripheral units can be turned off to further reduce the sleep power.

```
      powerc = 0x00F0   /* Turn off peripherals, core running with PLL */
sleep:a0 = 0x8000       /* Set alf register in cache loop if running from */
      do 1 {            /* external memory with >1 wait-state */
      alf = a0          /* Stop internal processor clock, interrupt circuits */
      nop               /* active  */
      }
      nop               /* Needed for bedtime execution. Only sleep power plus PLL */
      nop               /* power consumed here.... Interrupt wakes up the device */
cont: . . .             /* User code executes here */
      powerc = 0x0      /* Turn peripheral units back on */
```

## 4 Hardware Architecture (continued)

**Sleep with Slow Internal Clock, PLL Running**. In this case, the ring oscillator is selected to clock the processor before the device is put to sleep. This reduces power dissipation while waiting for an interrupt to continue program execution.

```
      powerc = 0x40F0   /* Turn off peripherals and select slow clock */
      2*nop             /* Wait for slow clock to take effect */
sleep:a0 = 0x8000       /* Set alf register in cache loop if running from */
      do 1 {            /* external memory with >1 wait-state */
      alf = a0          /* Stop internal processor clock, interrupt circuits */
      nop               /* active  */
      }
      nop               /* Needed for bedtime execution. Reduced sleep power, PLL */
      nop               /* power, and ring oscillator power consumed here... */
                        /* Interrupt wakes up the device */
cont: . . .             /* User code executes here */
      powerc = 0x00F0   /* Select high-speed PLL based clock */
      2*nop             /* Wait for it to take effect */
      powerc = 0x0000   /* Turn peripheral units back on */
```

**Sleep with Slow Internal Clock, PLL Disabled**. In this case, the slow clock must be selected first, and then the PLL must be disabled, since the PLL cannot run without the clock input circuitry being active.

```
      powerc = 0x40F0   /* Turn off peripherals and select slow clock */
      2*nop             /* Wait for slow clock to take effect */
      pllc = 0x29F2     /* Disable PLL (assume N = 1,M = 20, LF = 1001) */
sleep:a0 = 0x8000       /* Set alf register in cache loop if running from */
      do 1 {            /* external memory with >1 wait-state */
      alf = a0          /* Stop internal processor clock, interrupt circuits */
      nop               /* active  */
      }
      nop               /* Needed for bedtime execution. Reduced sleep power */
      nop               /* consumed here.... Interrupt wakes up device */
      pllc = 0xE9F2     /* Enable PLL, continue to run off slow clock */
      call pllwait      /* Loop to check for LOCK flag assertion */
cont: powerc = 0x00F0   /* Select high-speed PLL based clock */
      2*nop             /* Wait for it to take effect */
      powerc = 0x0000   /* Turn peripherals back on */
```

## 4 Hardware Architecture (continued)

**Software Stop, PLL Disabled**. In this case, all internal clocking is disabled. INT0, INT1, or RSTB can be used to reenable the clocks.

```
      powerc = 0x4000   /* SLOWCKI asserted */
      2*nop             /* Wait for slow clock to take effect */
      pllc = 0x29F2     /* Disable PLL (assume N = 1, M = 20, LF = 1001) */
      powerc = 0x5000   /* INT0EN asserted */
sopor:powerc = 0x7000   /* NOCK asserted, all clocks stop */
                        /* Minimum switching power consumed here */
      3*nop             /* Some nops will be needed */
                        /* INT0 pin clears NOCK field, clocking resumes */
cont: powerc = 0x4000   /* INTOEN cleared */
      pllc = 0xE9F2     /* Enable PLL, continue to run off slow clock */
      call pllwait      /* Loop to check for LOCK flag assertion */
      powerc = 0x0      /* Select high-speed PLL based clock */
      2*nop             /* Wait for it to take effect */
      ins = 0x0010      /* Clear the INT0 status bit */
```

An example subroutine for pllwait follows:

```
pllwait:    if lock return
            goto pllwait
```

# 5 Software Architecture

## 5.1 Instruction Set

The DSP1620 processor has seven types of instructions: multiply/ALU, special function, control, F3 ALU, BMU, cache, and data move. The multiply/ALU instructions are the primary instructions used to implement signal processing algorithms. Statements from this group can be combined to generate multiply/accumulate, logical, and other ALU functions and to transfer data between memory and registers in the data arithmetic unit. The special function instructions can be conditionally executed based on flags from the previous ALU or BMU operation, the condition of one of the counters, or the value of a pseudorandom bit in the DSP1620 device. Special function instructions perform shift, round, and complement functions. The F3 ALU instructions enrich the operations available on accumulators. The BMU instructions provide high-performance bit manipulation. The control instructions implement the goto and call commands. Control instructions can also be executed conditionally. Cache instructions are used to implement low-overhead loops, conserve program memory, and decrease the execution time of certain multiply/ALU instructions. Data move instructions are used to transfer data between memory and registers or between accumulators and registers. See the *DSP1620 Digital Signal Processor* Information Manual for a detailed description of the instruction set.

The following operators are used in describing the instruction set:

| | |
|---|---|
| * | 16 x 16-bit –> 32-bit multiplication **or** register-indirect addressing when used as a prefix to an address register **or** denotes direct addressing when used as a prefix to an immediate |
| + | 36-bit addition† |
| – | 36-bit subtraction† |
| >> | Arithmetic right shift |
| >>> | Logical right shift |
| << | Arithmetic left shift |
| <<< | Logical left shift |
| \| | 36-bit bitwise OR† |
| & | 36-bit bitwise AND† |
| ^ | 36-bit bitwise EXCLUSIVE OR† |
| : | Compound address swapping, accumulator shuffling |
| ~ | One's complement |

### Multiply/ALU Instructions

Note that the function statements and transfer statements in Table 25 are chosen independently. Any function statement (F1) can be combined with any transfer statement to form a valid multiply/ALU instruction. If either statement is not required, a single statement from either column constitutes a valid instruction. The number of cycles to execute the instruction is a function of the transfer column. (An instruction with no transfer statement executes in one instruction cycle.) Whenever PC, **pt**, or **rM** is used in the instruction and points to external memory, the programmed number of wait-states must be added to the instruction cycle count. All multiply/ALU instructions require one word of program memory. The no-operation (**nop**) instruction is a special case encoding of a multiply/ALU instruction and executes in one cycle. The assembly-language representation of a **nop** is either **nop** or a single semicolon.

A single-cycle squaring function is provided in DSP1620. By setting the X=Y= bit in the **auc** register, any instruction that loads the high half of the **y** register also loads the **x** register with the same value. A subsequent instruction to multiply the **x** register and **y** register results in the square of the value being placed in the **p** register. The instruction **a0** = **p** **p** = **x** * **y** **y** = * **r0**++ with the X=Y= bit set to one will read the value pointed to by **r0**, load it to both **x** and **y**, multiply the previously fetched value of **x** and **y**, and transfer the previous product to **a0**. A table of values pointed to by **r0** can thus be squared in a pipeline with one instruction cycle per each value. Multiply/ALU instructions that use **x** = X transfer statements (such as **a0** = **p** **p** = **x***y** **y** = ***r0**++ **x** = ***pt**++) are not recommended for squaring because **pt** will be incremented even though **x** is not loaded from the value pointed to by **pt**. Also, the same conflict wait occurrences from reading the same bank of internal memory or reading from external memory apply, since the X space fetch occurs (even though its value is not used).

† These are 36-bit operations. One operand is 36-bit data in an accumulator; the other operand can be 16, 32, or 36 bits.

## 5 Software Architecture (continued)

**Table 25. Multiply/ALU Instructions**

| Function Statement | |
|---|---|
| | p = x * y |
| aD = p | p = x * y |
| aD = aS + p | p = x * y |
| aD = aS – p | p = x * y |
| aD = p | |
| aD = aS + p | |
| aD = aS – p | |
| aD = y | |
| aD = aS + y | |
| aD = aS – y | |
| aD = aS & y | |
| aD = aS \| y | |
| aD = aS ^ y | |
| aS – y | |
| aS & y | |

| Transfer Statement* | | Cycles (Out/In Cache)† |
|---|---|---|
| y = Y | x = X | 2/1 |
| y = aT | x = X | 2/1 |
| y[l] = Y | | 1/1 |
| aT[l] = Y | | 1/1 |
| x = Y | | 1/1 |
| Y | | 1/1 |
| Y = y[l] | | 2/2 |
| Y = aT[l] | | 2/2 |
| Z:y | x = X | 2/2 |
| Z:y[l] | | 2/2 |
| Z:aT[l] | | 2/2 |

* The l in [ ] is an optional argument that specifies the low 16 bits of **aT** or **y**.
† Add cycles for:
1. When an external memory access is made in X or Y space and wait-states are programmed, add the number of wait-states.
2. If an X space access and a Y space access are made to the same bank of DPRAM in one instruction, add one cycle.

For transfer statements when loading the upper half of an accumulator, the lower half is cleared if the corresponding CLR bit in the **auc** register is zero. **auc** is cleared by reset.

**Table 26. Replacement Table for Multiply/ALU Instructions**

| Replace | Value | Meaning |
|---|---|---|
| aD, aS, aT | a0, a1 | One of two DAU accumulators. |
| X | *pt++, *pt++i | X memory space location pointed to by **pt**. **pt** is postmodified by +1 and **i**, respectively. |
| Y | *rM, *rM++, *rM--, rM++j | RAM location pointed to by **rM** (M = 0, 1, 2, 3). **rM** is postmodified by 0, +1, –1, or **j**, respectively. |
| Z | *rMzp, *rMpz, *rMm2, *rMjk | Read/Write compound addressing. **rM** (M = 0, 1, 2, 3) is used twice. First, postmodified by 0, +1, –1, or **j**, respectively; and, second, post-modified by +1, 0, +2, or **k**, respectively. |

## 5 Software Architecture (continued)

### Special Function Instructions

All forms of the special function require one word of program memory and execute in one instruction cycle. (If PC points to external memory, add programmed wait-states.)

| | |
|---|---|
| aD = aS >> 1<br>aD = aS >> 4<br>aD = aS >> 8<br>aD = aS >> 16 } | Arithmetic right shift (sign preserved) of 36-bit accumulators |

| | | |
|---|---|---|
| aD = aS | — | Load destination accumulator from source accumulator |
| aD = –aS | — | 2's complement |
| aD = ~aS* | — | 1's complement |
| aD = rnd(aS) | — | Round upper 20 bits of accumulator |
| aDh = aSh + 1 | — | Increment upper half of accumulator (lower half cleared) |
| aD = aS + 1 | — | Increment accumulator |
| aD = y | — | Load accumulator with 32-bit **y** register value with sign extend |
| aD = p | — | Load accumulator with 32-bit **p** register value with sign extend |

| | |
|---|---|
| aD = aS << 1<br>aD = aS << 4<br>aD = aS << 8<br>aD = aS << 16 } | Arithmetic left shift (sign not preserved) of the lower 32 bits of accumulators (upper 4 bits are sign-bit-extended from bit 31 at the completion of the shift) |

The above special functions can be conditionally executed, as in:

```
if CON instruction
```

and with an event counter

```
ifc CON instruction
```

which means:

```
if CON is true then
      c1 = c1 + 1
      instruction
      c2 = c1
else
      c1 = c1 + 1
```

The above special function statements can be executed unconditionally by writing them directly, e.g., **a0** = **a1**.

**Table 27. Replacement Table for Special Function Instructions**

| Replace | Value | Meaning |
|---|---|---|
| aD, aS | a0, a1 | One of two DAU accumulators. |
| CON | mi, pl, eq, ne, gt, le, lvs, lvc, mvs, mvc, c0ge, c0lt, c1ge, c1lt, heads, tails, true, false, allt, allf, somet, somef, oddp, evenp, mns1, nmns1, npint, njint, lock, ebusy, mioubusy | See Table 29 for definitions of mnemonics. |

* This function is not available for the DSP16A.

## 5 Software Architecture (continued)

### Control Instructions

All control instructions executed unconditionally execute in two cycles, except **icall** which takes three cycles. Control instructions executed conditionally execute in three instruction cycles. (If PC, **pt**, or **pr** point to external memory, add programmed wait-states.) Control instructions executed unconditionally require one word of program memory, while control instructions executed conditionally require two words. Control instructions cannot be executed from the cache.

```
goto JA*
goto pt
call JA*
call pt
icall†
return      (goto pr)
ireturn     (goto pi)
```

The above control instructions, with the exception of **ireturn** and **icall**, can be conditionally executed. For example:

```
if le goto 0x0345
```

**Table 28. Replacement Table for Control Instructions**

| Replace | Value | Meaning |
|---|---|---|
| CON | mi, pl, eq, ne, gt, le, lvs, lvc, mvs, mvc, c0ge, c0lt, c1ge, c1lt, heads, tails, true, false, allt, allf, somet, somef, oddp, evenp, mns1, nmns1, npint, njint, lock, ebusy, mioubusy | See Table 29 for definitions of mnemonics. |
| JA | 12-bit value | Least significant 12 bits of absolute address within the same 4 Kwords memory section. |

* The **goto JA** and **call JA** instructions should not be placed in the last or next-to-last instruction before the boundary of a 4 Kwords page. If the **goto** or **call** is placed there, the program counter will have incremented to the next page and the jump will be to the next page, rather than to the desired current page.

† The **icall** instruction is reserved for development system use.

## 5 Software Architecture (continued)

### Conditional Mnemonics (Flags)

Table 29 lists mnemonics used in conditional execution of special function and control instructions.

**Table 29. DSP1620 Conditional Mnemonics**

| Test | Meaning | Test | Meaning |
|---|---|---|---|
| pl | Result is nonnegative (sign bit is bit 35) (≥0). | mi | Result is negative (<0). |
| eq | Result is equal to 0 (=0). | ne | Result is not equal to 0 (≠0). |
| gt | Result is greater than 0 (>0). | le | Result is less than or equal to 0 (≤0). |
| lvs | Logical overflow set.* | lvc | Logical overflow clear. |
| mvs | Mathematical overflow set.† | mvc | Mathematical overflow clear. |
| c0ge | Counter 0 greater than or equal to 0. | c0lt | Counter 0 less than 0. |
| c1ge | Counter 1 greater than or equal to 0. | c1lt | Counter 1 less than 0. |
| heads | Pseudorandom sequence bit set. | tails | Pseudorandom sequence bit clear. |
| true | The condition is always satisfied in an if instruction. | false | The condition is never satisfied in an if instruction. |
| allt | All true, all BIO input bits tested compared successfully. | allf | All false, no BIO input bits tested compared successfully. |
| somet | Some true, some BIO input bits tested compared successfully. | somef | Some false, some BIO input bits tested did not compare successfully. |
| oddp | Odd parity, from BMU operation. | evenp | Even parity, from BMU operation. |
| mns1 | Minus 1, result of BMU operation. | nmns1 | Not minus 1, result of BMU operation. |
| npint | Not PINT, used by hardware development system. | njint | Not JINT, used by hardware development system. |
| lock | The PLL has achieved lock and is stable. | ebusy | ECCP busy. |
| mioubusy | MIOU0, MIOU1, or both have unfinished output operations pending. | — | — |

* Result is not representable in the 36-bit accumulators (36-bit overflow).
† Bits 35—31 are not the same (32-bit overflow).

Notes:
Testing the state of the counters (**c0** or **c1**) automatically increments the counter by one.

The heads or tails condition is determined by a randomly set or cleared bit, respectively. The bit is randomly set with a probability of 0.5. A random rounding function can be implemented with either heads or tails. The random bit is generated by a ten-stage pseudorandom sequence generator (PSG) that is updated after either a heads or tails test. The pseudorandom sequence can be reset by writing any value to the **pi** register, except during an interrupt service routine (ISR). While in an ISR, writing to the **pi** register updates the register and does not reset the PSG. If not in an ISR, writing to the **pi** register resets the PSG. (The **pi** register is updated, but is written with the contents of the PC on the next instruction.) **Interrupts must be disabled when writing to the pi register.** If an interrupt is taken after the **pi** write, but before **pi** is updated with the PC value, the **ireturn** instruction will not return to the correct location. If the RAND bit in the **auc** register is set, however, writing the **pi** register never resets the PSG.

## 5 Software Architecture (continued)

### F3 ALU Instructions

These instructions are implemented in the DSP1600 core. They allow accumulator two-operand operations with either another accumulator, the **p** register, or a 16-bit immediate operand (IM16). The result is placed in a destination accumulator that can be independently specified. All operations are done with the full 36 bits. For the accumulator with accumulator operations, both inputs are 36 bits. For the accumulator with **p** register operations, the **p** register is sign-extended into bits 35—32 before the operation. For the accumulator high with immediate operations, the immediate is sign-extended into bits 35—32 and the lower bits 15—0 are filled with zeros, except for the AND operation, for which they are filled with ones. These conventions allow the user to do operations with 32-bit immediates by programming two consecutive 16-bit immediate operations. The F3 ALU instructions are shown in Table 30.

**Table 30. F3 ALU Instructions**

The F3 ALU instructions that do not have a destination accumulator are used to set flags for conditional operations, i.e., bit test operations.

| **F3 ALU Instructions*** | |
|---|---|
| Cachable (one-cycle) | Not Cachable (two-cycle)† |
| aD = aS + aT | aD = aSh + IM16 |
| aD = aS – aT | aD = aSh – IM16 |
| aD = aS & aT | aD = aSh & IM16 |
| aD = aS \| aT | aD = aSh \| IM16 |
| aD = aS ^ aT | aD = aSh ^ IM16 |
| aS – aT | aSh – IM16 |
| aS & aT | aSh & IM16 |
| aD = aS + p | aD = aSl + IM16 |
| aD = aS – p | aD = aSl – IM16 |
| aD = aS & p | aD = aSl & IM16 |
| aD = aS \| p | aD = aSl \| IM16 |
| aD = aS ^ p | aD = aSl ^ IM16 |
| aS – p | aSl – IM16 |
| aS & p | aSl & IM16 |

* If PC points to external memory, add programmed wait-states.
† The **h** and **l** are required notation in these instructions.

### BMU Instructions

The bit manipulation unit in the DSP1620 provides a set of efficient bit manipulation operations on accumulators. It contains four auxiliary registers, **ar<0—3>** (**arM**, **M** = 0, 1, 2, 3), two alternate accumulators (**aa0**—**aa1**), that can be shuffled with the working set, and four flags (oddp, evenp, mns1, and nmns1). The flags are testable by conditional instructions and can be read and written via bits 4—7 of the **alf** register. The BMU also sets the LMI, LEQ, LLV, and LMV flags in the **psw** register:

- LMI = 1 if negative (i.e., bit 35 = 1)
- LEQ = 1 if zero (i.e., bits 35—0 are 0)
- LLV = 1 if (a) 36-bit overflow, or if (b) illegal shift on field width/offset condition
- LMV = 1 if bits 31—35 are not the same (32-bit overflow)

The BMU instructions and cycle times follow. (If PC points to external memory, add programmed wait-states.) All BMU instructions require 1 word of program memory unless otherwise noted. Please refer to the *DSP1620 Digital Signal Processor* Information Manual for further discussion of the BMU instructions.

## 5 Software Architecture (continued)

- **Barrel Shifter**

| | |
|---|---|
| aD = aS >> IM16 | Arithmetic right shift by immediate (36-bit, sign filled in); 2-cycle, 2-word. |
| aD = aS >> arM | Arithmetic right shift by **arM** (36-bit, sign filled in); 1-cycle. |
| aD = $\overline{\text{aS}}$ >> aS | Arithmetic right shift by aS (36-bit, sign filled in); 2-cycle. |
| aD = aS >>> IM16 | Logical right shift by immediate (32-bit shift, 0s filled in); 2-cycle, 2-word. |
| aD = aS >>> arM | Logical right shift by **arM** (32-bit shift, 0s filled in); 1-cycle. |
| aD = $\overline{\text{aS}}$ >>> aS | Logical right shift by aS (32-bit shift, 0s filled in); 2-cycle. |
| aD = aS <<< IM16 | Arithmetic left shift* by immediate (36-bit shift, 0s filled in); 2-cycle, 2-word. |
| aD = aS <<< arM | Arithmetic left shift* by **arM** (36-bit shift, 0s filled in); 1-cycle. |
| aD = $\overline{\text{aS}}$ << aS | Arithmetic left shift* by aS (36-bit shift, 0s filled in); 2-cycle. |
| aD = aS <<< IM16 | Logical left shift by immediate (36-bit shift, 0s filled in); 2-cycle, 2-word. |
| aD = aS <<< arM | Logical left shift by **arM** (36-bit shift, 0s filled in); 1-cycle. |
| aD = $\overline{\text{aS}}$ <<< aS | Logical left shift by aS (36-bit shift, 0s filled in); 2-cycle. |

- **Normalization and Exponent Computation**

| | |
|---|---|
| aD = exp(aS) | Detect the number of redundant sign bits in accumulator; 1-cycle. |
| aD = norm(aS, arM) | Normalize aS with respect to bit 31, with exponent in **arM**; 1-cycle. |

- **Bit-Field Extraction and Insertion**

| | |
|---|---|
| aD = extracts(aS, IM16) | Extraction with sign extension, field specified as immediate; 2-cycle, 2-word. |
| aD = extracts(aS, arM) | Extraction with sign extension, field specified in **arM**; 1-cycle. |
| aD = extractz(aS, IM16) | Extraction with zero extension, field specified as immediate; 2-cycle, 2-word. |
| aD = extractz(aS, arM) | Extraction with zero extension, field specified in **arM**; 1-cycle. |
| aD = insert(aS, IM16) | Bit-field insertion, field specified as immediate; 2-cycle, 2-word. |
| aD = insert(aS, arM) | Bit-field insertion, field specified in **arM**; 2-cycle. |

**Note**: The bit field to be inserted or extracted is specified as follows. The width (in bits) of the field is the upper byte of the operand (immediate or **arM**), and the offset from the LSB is in the lower byte.

- **Alternate Accumulator Set**

| | |
|---|---|
| aD = aS:aa0 | Shuffle accumulators with alternate accumulator 0 (**aa0**); 1-cycle. |
| aD = aS:aa1 | Shuffle accumulators with alternate accumulator 1 (**aa1**); 1-cycle. |

**Note**: The alternate accumulator gets what was in aS. aD gets what was in the alternate accumulator.

**Table 31. Replacement Table for F3 ALU Instructions and BMU Instructions**

| Replace | Value | Meaning |
|---|---|---|
| aD, aT, aS | a0 or a1 | One of the two accumulators. |
| IM16 | immediate | 16-bit data, sign-, zero-, or one-extended as appropriate. |
| arM | ar<0—3> | One of the auxiliary BMU registers. |

* Not the same as the special function arithmetic left shift. Here, the guard bits in the destination accumulator are shifted into, not sign-extended.

## 5 Software Architecture (continued)

### Cache Instructions

Cache instructions require one word of program memory. The **do** instruction executes in one instruction cycle, and the **redo** instruction executes in two instruction cycles. (If PC points to external memory, add programmed wait-states.) Control instructions and long immediate values cannot be stored inside the cache. The instruction formats are as follows:

```
do K {
      instr1
      instr2
      .
      .
      .
      instrN
      }

redo K
```

**Table 32. Replacement Table for Cache Instructions**

| Replace | Instruction Encoding | Meaning |
|---|---|---|
| K | cloop* | Number of times the instructions are to be executed taken from bits 0—6 of the **cloop** register. |
| | 1 to 127 | Number of times the instructions to be executed are encoded in the instruction. |
| N | 1 to 15 | 1 to 15 instructions can be included. |

* The assembly-language statement, **do cloop** (or **redo cloop**), is used to specify that the number of iterations is to be taken from the **cloop** register. K is encoded as 0 in the instruction encoding to select **cloop**.

When the cache is used to execute a block of instructions, the cycle timings of the instructions are as follows:

1. In the first pass, the instructions are fetched from program memory and the cycle times are the normal out-of-cache values, except for the last instruction in the block of N instructions. This instruction executes in two cycles.
2. During pass two through pass K – 1, each instruction is fetched from cache and the in-cache timings apply.
3. During the last (Kth) pass, the block of instructions is fetched from cache and the in-cache timings apply, except that the timing of the last instruction is the same as if it were out-of-cache.
4. If any of the instructions access external memory, programmed wait-states must be added to the cycle counts.

The **redo** instruction treats the instructions currently in the cache memory as another loop to be executed K times. Using the **redo** instruction, instructions are reexecuted from the cache without reloading the cache.

The number of iterations, K, for a **do** or **redo** can be set at run time by first moving the number of iterations into the **cloop** register (7 bits unsigned), and then issuing the **do cloop** or **redo cloop**. At the completion of the loop, the value of **cloop** is decremented to 0; hence, **cloop** needs to be written before each **do cloop** or **redo cloop**.

## 5 Software Architecture (continued)

### Data Move Instructions

Data move instructions normally execute in two instruction cycles. (If PC or **rM** point to external memory, any programmed wait-states must be added. In addition, if PC and **rM** point to the same bank of DPRAM, then one cycle must be added.) Immediate data move instructions require two words of program memory; all other data move instructions require only one word. The only exception to these statements is a special case immediate load (short immediate) instruction. If a YAAU register is loaded with a 9-bit short immediate value, the instruction requires only one word of memory and executes in one instruction cycle. All data move instructions, except those doing long immediate loads, can be executed from within the cache. The data move instructions are as follows:

R = IM16

aT[l] = R

SR = IM9

Y = R

R = Y

Z : R

R = aS[l]

DR = *(OFFSET)

*(OFFSET) = DR

**Table 33. Replacement Table for Data Move Instructions**

| Replace | Value | Meaning |
|---|---|---|
| R | Any of the registers in Table 66 | — |
| DR | r<0—3>, a0[l], a1[l], y[l], p, pl, x, pt, pr, psw | Subset of registers accessible with direct addressing. |
| aS, aT | a0, a1 | High half of accumulator. |
| Y | *rM, *rM++, *rM--, *rM++j | Same as in multiply/ALU instructions. |
| Z | *rMzp, *rMpz, *rMm2, *rMjk | Same as in multiply/ALU instructions. |
| IM16 | 16-bit value | Long immediate data. |
| IM9 | 9-bit value | Short immediate data for YAAU registers. |
| OFFSET | 5-bit value from instruction<br>11-bit value in base register | Value in bits [15:5] of **ybase** register form the 11 most significant bits of the base address. The 5-bit offset is concatenated to this to form a 16-bit address. |
| SR | r<0—3>, rb, re, j, k | Subset of registers for short immediate. |

Notes:
**sioc**, **sioc2**, **tdms**, **tdms2**, **srta**, and **srta2** registers are not readable.

When signed registers less than 16 bits wide (**c0**, **c1**, **c2**) are read, their contents are sign-extended to 16 bits. When unsigned registers less than 16 bits wide are read, their contents are zero-extended to 16 bits.

Loading an accumulator with a data move instruction does not affect the flags.

## 5 Software Architecture (continued)

### 5.2 Register Settings

Tables 34 through 54 describe the programmable registers of the DSP1620 device. Table 55 describes the register settings after reset.

Note that the following abbreviations are used in the tables:

- x = don't care
- R = read only
- W = read/write

The reserved (Rsvd) bits in the tables should always be written with zeros to make the program compatible with future chip versions.

**Table 34. alf Register**

| **Bit** | 15 | 14 | 13—10 | 9—0 |
|---|---|---|---|---|
| **Field** | AWAIT | LOWPR | Reserved | FLAGS |

| **Field** | **Value** | **Action** |
|---|---|---|
| AWAIT | 0<br>1 | Normal operation.<br>Power-saving standby mode or standard sleep enabled. |
| LOWPR | 0<br>1 | The internal DPRAM is addressed beginning at 0x8000 in X space.<br>The internal DPRAM is addressed beginning at 0x0000 in X space. |
| Reserved | — | Reserved—write with zero. |
| FLAGS | — | See table below. |

| **Bit** | **Flag** | **Use** |
|---|---|---|
| 9 | mioubusy* | MIOU0 or MIOU1 have output work pending |
| 8 | ebusy* | ECCP BUSY |
| 7 | nmns1 | NOT-MINUS-ONE from BMU |
| 6 | mns1 | MINUS-ONE from BMU |
| 5 | evenp | EVEN PARITY from BMU |
| 4 | oddp | ODD PARITY from BMU |
| 3 | somef | SOME FALSE from BIO |
| 2 | somet | SOME TRUE from BIO |
| 1 | allf | ALL FALSE from BIO |
| 0 | allt | ALL TRUE from BIO |

* The ebusy and mioubusy flags cannot be written by the user.

## 5 Software Architecture (continued)

**Table 35. auc — Arithmetic Unit Control Register***

| **Bit** | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| **Field** | RAND | X=Y= | CLR | | | SAT | | ALIGN | |

| Field | Value | Description |
|---|---|---|
| RAND | 0 | Pseudorandom sequence generator (PSG) reset by writing the **pi** register only outside an interrupt service routine. |
| | 1 | PSG never reset by writing the **pi** register. |
| X=Y= | 0 | Normal operation. |
| | 1 | All instructions that load the high half of the **y** register also load the **x** register, allowing single-cycle squaring with **p** = **x** * **y**. |
| CLR | 1xx | Clearing **yl** is disabled (enabled when 0). |
| | x1x | Clearing **a1l** is disabled (enabled when 0). |
| | xx1 | Clearing **a0l** is disabled (enabled when 0). |
| SAT | 1x | **a1** saturation on overflow is disabled (enabled when 0). |
| | x1 | **a0** saturation on overflow is disabled (enabled when 0). |
| ALIGN | 00 | **a0**, **a1** ← p. |
| | 01 | **a0**, **a1** ← p/4. |
| | 10 | **a0**, **a1** ← p x 4 (and zeros written to the two LSBs). |
| | 11 | **a0**, **a1** ← p x 2 (and zero written to the LSB). |

* The **auc** register is 16 bits. The upper 7 bits [15:9] are always zero when read and should always be written with zeros to make the program compatible with future chip versions. The **auc** register is cleared at reset.

## 5 Software Architecture (continued)

**Table 36. cbit — BIO Control Register**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | 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 |
| 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 |

* $0 \le n \le 7$.

**Table 37. sbit — BIO Status Register**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | DIREC[7:0] | | | | | | | | VALUE[7:0] | | | | | | | |

| Field | Value | Description |
|---|---|---|
| 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). |
| 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. |

* $0 \le n \le 7$.

**Table 38. ID — JTAG Identification Register**

| Bit | 31—30 | 29—28 | 27—19 | 18—12 | 11—0 |
|---|---|---|---|---|---|
| Field | RSVD 0 | 0x3 | RSVD 1 | PART ID | 0x03B |

| Field | Value | Features |
|---|---|---|
| RSVD 0* | 00 | — |
| RSVD 1* | 000000000 | — |
| PART ID | 0x22 | DSP1620 |

* The values in RSVD 0 and RSVD 1 are subject to change; user applications should not depend on the values in these fields.

## 5 Software Architecture (continued)

**Table 39. inc — Interrupt Control Register***

| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7—4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Field** | JINT† | Rsvd | EOVF | EREADY | WAKEUP | MOBE0 | MIBF0 | TIME | INT[3:0] | MIBF1 | MOBE1 | OBE | IBF |

* Encoding: A 0 disables an interrupt; a 1 enables an interrupt. After reset, all interrupts are disabled.
† JINT is a JTAG interrupt and is controlled by the HDS. It can be made unmaskable by the Lucent Technologies development system tools.

**Table 40. ins — Interrupt Status Register***

| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7—4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Field** | JINT | Rsvd | EOVF | EREADY | WAKEUP | MOBE0 | MIBF0 | TIME | INT[3:0] | MIBF1 | MOBE1 | OBE | IBF |

* Encoding: A 0 indicates no interrupt is pending. A 1 indicates an interrupt has been recognized and is pending or being serviced. If a 1 is written to bits 4—7, 8, 11, 12, or 13 of **ins**, the corresponding interrupt is cleared.

## 5 Software Architecture (continued)

**Table 41. ioc — I/O Configuration Register**

| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8—7 | 6 | 5 | 4—0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Field** | Rsvd | EXTROM | CKO2 | EBIOH | WEROM | Rsvd | SIOLBC | CKO[1:0] | Rsvd | RWNADV | DENB[4:0] |

| Field | Description |
|---|---|
| Rsvd | Reserved—write with zero. |
| EXTROM | Selects between XMAP3/XMAP4 and YMAP3/YMAP4. (See Table 5 and Table 6.) |
| CKO2 | CKO configuration (see CKO Configuration below). |
| EBIOH | If 1, enables high half of BIO, IOBIT[4:7], and disables VEC[3:0] from pins. |
| WEROM | Selects YMAP3 or YMAP4. (See Table 6.) |
| Rsvd | Reserved—write with zero. |
| SIOLBC | If 1, DO1 and DO2 looped back to DI1 and DI2. |
| CKO[1:0] | CKO configuration (see CKO Configuration below). |
| Rsvd | Reserved—write with zero. |
| RWNADV | If 0, delay RWN with respect to (undelayed) enables.<br>If 1, align RWN with respect to (undelayed) enables. |
| DENB4 | If 1, delay ERAMX. |
| DENB3 | If 1, delay EROM. |
| DENB2 | If 1, delay ERAMHI. |
| DENB1 | If 1, delay I/O. |
| DENB0 | If 1, delay ERAMLO. |

**CKO Configuration**

| CKO2 | CKO1 | CKO0 | CKO Output | | Description |
|---|---|---|---|---|---|
| | | | 1X | PLL | |
| 0 | 0 | 0 | CKI | CKI x M/(2N) | Free-running clock.*, † |
| 0 | 0 | 1 | CKI/(1 + W) | CKI x (M/(2N))/[1 + W] | Wait-stated clock.*, †, ‡ |
| 0 | 1 | 0 | 1 | 1 | Held high. |
| 0 | 1 | 1 | 0 | 0 | Held low. |
| 1 | 0 | 0 | CKI | CKI | Output of CKI buffer. |
| 1 | 0 | 1 | CKI/(1 + W) | CKI x (M/(2N))/[1 + W] | Sequenced, wait-stated clock.*, †, ‡, § |
| 1 | 1 | 0 | Reserved | | |
| 1 | 1 | 1 | | | |

* The phase of CKI is synchronized by the rising edge of RSTB.
† When SLOWCKI is enabled in the **powerc** register, these options reflect the low-speed internal ring oscillator. W is the number of wait-states (see Table 42). M and N are defined in the phase-locked loop control (**pllc**) register (see Table 44).
‡ The wait-stated clock reflects the internal instruction cycle and can be stretched based on the **mwait** register setting (see Table 42). During sequenced external memory accesses, it completes one cycle.
§ The sequenced wait-stated clock completes two cycles during a sequenced external memory access and can be stretched based on the **mwait** register setting (see Table 42).

**Table 42. mwait — EMI Configuration Register***

| **Bit** | 15—12 | 11—8 | 7—4 | 3—0 |
|---|---|---|---|---|
| **Field** | EROM[3:0] | ERAMHI[3:0] | I/O[3:0] | ERAMLO[3:0] |

* If the EXM pin is high and the INT1 is low upon reset, the **mwait** register is initialized to all 1s (15 wait-states for all external memory). Otherwise, the **mwait** register is initialized to all 0s (0 wait-states) upon reset.

## 5 Software Architecture (continued)

**Table 43. PHIFC — PHIF16 Control Register***

| **Bit** | 15—8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| **Field** | Rsvd | PCFIG | PSOBEF | PFLAGSEL | PFLAG | PBSELF | PSTRB | PSTROBE | PMODE |

| **Field** | **Value** | **Description** (See Table 9 and Table 10.) |
|---|---|---|
| Rsvd | — | Reserved—write with zero. |
| PCFIG | 0<br>1 | 8-bit external bus configuration. PB[15:8] are 3-stated.<br>16-bit external bus configuration. |
| PSOBEF | 0<br>1 | Normal.<br>POBE flag as read through PSTAT register is active-low. |
| PFLAGSEL | 0<br>1 | Normal.<br>PIBF flag ORed with POBE flag and output on PIBF pin; POBE pin unchanged (output buffer empty). |
| PFLAG | 0<br>1 | PIBF and POBE pins active-high.<br>PIBF and POBE pins active-low. |
| PBSELF | 0<br><br><br>1 | If PMODE = 0, PBSEL pin = 0 -> PDX low byte.<br>If PMODE = 1, PBSEL pin = 0 -> PDX low byte.<br>If PMODE = 1, PBSEL pin = 1 -> PDX high byte.<br>If PMODE = 0, PBSEL pin = 1 -> PDX low byte.<br>If PMODE = 1, PBSEL pin = 0 -> PDX high byte.<br>If PMODE = 1, PBSEL pin = 1 -> PDX low byte. |
| PSTRB | 0<br>1 | When PSTROBE = 1, PODS pin (PDS) active-low.<br>When PSTROBE = 1, PODS pin (PDS) active-high. |
| PSTROBE | 0<br>1 | *Intel* protocol: PIDS and PODS data strobes.<br>*Motorola* protocol: PRWN and PDS data strobes. |
| PMODE | 0<br>0<br>1<br>1 | If 8-bit external configuration, 8-bit logical data transfers.<br>If 16-bit external configuration, preserve high and low byte positions.<br>If 8-bit external configuration, 16-bit logical data transfers.<br>If 16-bit external configuration, swap high and low byte positions. |

* **PHIFC** must be programmed through MIOU1.

**Table 44. pllc — Phase-Locked Loop Control Register**

| **Bit** | 15 | 14 | 13 | 12 | 11—8 | 7—5 | 4—0 |
|---|---|---|---|---|---|---|---|
| **Field** | PLLEN | PLLSEL | ICP | SEL5V | LF[3:0] | Nbits[2:0] | Mbits[4:0] |

| **Field** | **Value** | **Description** |
|---|---|---|
| PLLEN | 0<br>1 | PLL powered down.<br>PLL powered up. |
| PLLSEL | 0<br>1 | DSP internal clock taken directly from CKI.<br>DSP internal clock taken from PLL. |
| ICP | — | Charge pump current selection (see Table 78 for proper value). |
| SEL5V | 0<br>1 | 3 V operation (see Table 78 for proper value).<br>5 V operation (see Table 78 for proper value). |
| LF[3:0] | — | Loop filter setting (see Table 78 for proper value). |
| Nbits[2:0] | — | Encodes N, $1 \le N \le 8$, where N = Nbits[2:0] + 2, unless Nbits[2:0] = 111, then N = 1. |
| Mbits[4:0] | — | Encodes M, $2 \le M \le 24$, where M = Mbits[4:0] + 2, $f_{\text{internal clock}} = f_{\text{CKI}} \times (M/(2N))$. |

## 5 Software Architecture (continued)

**Table 45. powerc — Power Control Register**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9—8 | 7 | 6 | 5 | 4 | 3—1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Field** | Rsvd | SLOWCKI | NOCK | INT0EN | Rsvd | INT1EN | Rsvd | SIO1DIS | SSIODIS | PHIFDIS | TIMERDIS | Rsvd | ECCPDIS |

| Field | Description |
|---|---|
| Rsvd* | Reserved—write with zero. |
| SLOWCKI | 1 = select ring oscillator clock (internal slow clock). |
| NOCK | 1 = disable internal processor clock. |
| INT0EN | 1 = INT0 clears NOCK field. |
| Rsvd* | Reserved—write with zero. |
| INT1EN | 1 = INT1 clears NOCK field. |
| Rsvd* | Reserved—write with zero. |
| SIO1DIS | 1 = disable SIO. |
| SSIODIS | 1 = disable SSIO and MIOU0. |
| PHIFDIS | 1 = disable PHIF16 and MIOU1. |
| TIMERDIS | 1 = disable timer. |
| Rsvd* | Reserved—write with zero. |
| ECCPDIS | 1 = disable ECCP. |

* The reserved (Rsvd) bits should always be written with zeros to make the program compatible with future chip versions.

**Table 46. psw — Processor Status Word Register**

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Field** | DAU FLAGS | | | | X | X | a1[V] | a1[35:32] | | | | a0[V] | a0[35:32] | | | |

| Field | Value | Description |
|---|---|---|
| DAU FLAGS* | Wxxx | LMI—logical minus when set (bit 35 = 1). |
| | xWxx | LEQ—logical equal when set (bit [35:0] = 0). |
| | xxWx | LLV—logical overflow when set. |
| | xxxW | LMV—mathematical overflow when set. |
| a1[V] | W | Accumulator 1 (**a1**) overflow when set. |
| a1[35:32] | Wxxx | Accumulator 1 (**a1**) bit 35. |
| | xWxx | Accumulator 1 (**a1**) bit 34. |
| | xxWx | Accumulator 1 (**a1**) bit 33. |
| | xxxW | Accumulator 1 (**a1**) bit 32. |
| a0[V] | W | Accumulator 0 (**a0**) overflow when set. |
| a0[35:32] | Wxxx | Accumulator 0 (**a0**) bit 35. |
| | xWxx | Accumulator 0 (**a0**) bit 34. |
| | xxWx | Accumulator 0 (**a0**) bit 33. |
| | xxxW | Accumulator 0 (**a0**) bit 32. |

* The DAU flags can be set by either BMU or DAU operations.

**Table 47. saddx — Multiprocessor Serial Address/Protocol Register**

| Bit Field | 15—8 | 7—0 |
|---|---|---|
| **Write** | X | Write Protocol Field [7:0] |
| **Read** | Read Protocol Field [7:0] | 0 |

## 5 Software Architecture (continued)

**Table 48. sbit — BIO Status Register**

| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Field** | DIREC[7:0] | | | | | | | | VALUE[7:0] | | | | | | | |

| **Field** | **Value** | **Description** |
|---|---|---|
| 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). |
| 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. |

* $0 \leq n \leq 7$.

**Table 49. cbit — BIO Control Register**

| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Field** | 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 |
| 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 |

* $0 \leq n \leq 7$.

## 5 Software Architecture (continued)

**Table 50. sioc — Serial I/O Control Registers**

| **Bit** | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Field** | DODLY1 | LD1 | CLK1 | | MSB1 | OLD1 | ILD1 | OCK1 | ICK1 | OLEN1 | ILEN1 |

| Field | Value | Description |
|---|---|---|
| DODLY1 | 0<br>1 | DO1 changes on the rising edge of OCK1.<br>DO1 changes on the falling edge of OCK1. The delay in driving DO1 increases the hold time on DO1 by half a cycle of OCK1. |
| LD1 | 0<br>1 | In active mode, ILD1 and/or OLD1 = ICK1 ÷ 16, active SYNC1 = ICK1 ÷ [128 or 256*].<br>In active mode, ILD1 and/or OLD1 = OCK1 ÷ 16, active SYNC1 = OCK1 ÷ [128 or 256*]. |
| CLK1 | 00<br>01<br>10<br>11 | Active clock = CKO ÷ 2 (1X).<br>Active clock = CKO ÷ 6 (1X).<br>Active clock = CKO ÷ 8 (1X).<br>Active clock = CKO ÷ 10 (1X). |
| MSB1 | 0<br>1 | LSB first.<br>MSB first. |
| OLD1 | 0<br>1 | OLD1 is an input (passive mode).<br>OLD1 is an output (active mode). |
| ILD1 | 0<br>1 | ILD1 is an input (passive mode).<br>ILD1 is an output (active mode). |
| OCK1 | 0<br>1 | OCK1 is an input (passive mode).<br>OCK1 is an output (active mode). |
| ICK1 | 0<br>1 | ICK1 is an input (passive mode).<br>ICK1 is an output (active mode). |
| OLEN1 | 0<br>1 | 16-bit output.<br>8-bit output. |
| ILEN1 | 0<br>1 | 16-bit input.<br>8-bit input. |

* See **tdms** register, SYNC field.

## 5 Software Architecture (continued)

**Table 51. srta — Serial Receive/Transmit Address Register**

| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Field** | RECEIVE ADDRESS | | | | | | | | TRANSMIT ADDRESS | | | | | | | |

| **Field** | **Value** | **Description** |
|---|---|---|
| RECEIVE ADDRESS | 1xxxxxxx | Receive address 7. |
| | x1xxxxxx | Receive address 6. |
| | xx1xxxxx | Receive address 5. |
| | xxx1xxxx | Receive address 4. |
| | xxxx1xxx | Receive address 3. |
| | xxxxx1xx | Receive address 2. |
| | xxxxxx1x | Receive address 1. |
| | xxxxxxx1 | Receive address 0. |
| TRANSMIT ADDRESS | 1xxxxxxx | Transmit address 7. |
| | x1xxxxxx | Transmit address 6. |
| | xx1xxxxx | Transmit address 5. |
| | xxx1xxxx | Transmit address 4. |
| | xxxx1xxx | Transmit address 3. |
| | xxxxx1xx | Transmit address 2. |
| | xxxxxx1x | Transmit address 1. |
| | xxxxxxx1 | Transmit address 0. |

## 5 Software Architecture (continued)

**Table 52. SSIOC — Simple Serial I/O Control Registers**

| **Bit** | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Field** | DODLY2 | LD2 | CLK2 | | MSB2 | OLD2 | ILD2 | OCK2 | ICK2 | OLEN2 | ILEN2 |

| **Field**[*] | **Value** | **Description** |
|---|---|---|
| DODLY2 | 0<br>1 | DO2 changes on the rising edge of OCK2.<br>DO2 changes on the falling edge of OCK2. The delay in driving DO2 increases the hold time on DO2 by half a cycle of OCK2. |
| LD2 | 0<br>1 | In active mode, ILD2 and/or OLD2 = ICK2 ÷ 16.<br>In active mode, ILD2 and/or OLD2 = OCK2 ÷ 16. |
| CLK2 | 00<br>01<br>10<br>11 | Active clock = CKO ÷ 2 (1X).<br>Active clock = CKO ÷ 6 (1X).<br>Active clock = CKO ÷ 8 (1X).<br>Active clock = CKO ÷ 10 (1X). |
| MSB2 | 0<br>1 | LSB first.<br>MSB first. |
| OLD2 | 0<br>1 | OLD2 is an input (passive mode).<br>OLD2 is an output (active mode). |
| ILD2 | 0<br>1 | ILD2 is an input (passive mode).<br>ILD2 is an output (active mode). |
| OCK2 | 0<br>1 | OCK2 is an input (passive mode).<br>OCK2 is an output (active mode). |
| ICK2 | 0<br>1 | ICK2 is an input (passive mode).<br>ICK2 is an output (active mode). |
| OLEN2 | 0<br>1 | 16-bit output.<br>8-bit output. |
| ILEN2 | 0<br>1 | 16-bit input.<br>8-bit input. |

* The field definitions for the **SSIOC** are identical to those of the **sioc** register (see Table 50). The **SSIOC** register is accessible through MIOU0 only.

## 5 Software Architecture (continued))

**Table 53. tdms — Time-Division Multiplex Slot Register**

| **Bit** | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Field** | SYNCSP | MODE | TRANSMIT SLOT | | | | | | | SYNC |

| **Field** | **Value** | **Description** |
|---|---|---|
| SYNCSP* | 0† | SYNC1 = ICK1/128 if LD = 0. |
| | | SYNC1 = OCK1/128 if LD = 1. |
| | | SYNC1 = ICK1/256 if LD = 0. |
| | 1 | SYNC1 = OCK1/256 if LD = 1. |
| MODE | 0 | Multiprocessor mode off; DOEN1 is an input (passive mode). |
| | 1 | Multiprocessor mode on; DOEN1 is an output (active mode). |
| TRANSMIT SLOT | 1xxxxxx | Transmit slot 7. |
| | x1xxxxx | Transmit slot 6. |
| | xx1xxxx | Transmit slot 5. |
| | xxx1xxx | Transmit slot 4. |
| | xxxx1xx | Transmit slot 3. |
| | xxxxx1x | Transmit slot 2. |
| | xxxxxx1 | Transmit slot 1. |
| SYNC | 1 | Transmit slot 0, SYNC1 is an output (active mode). |
| | 0 | SYNC1 is an input (passive mode). |

* See **sioc** register, LD1 field.
† Select this mode when in multiprocessor mode.

**Table 54. timerc — Timer Control Register**

| **Bit** | 15—7 | 6 | 5 | 4 | 3—0 |
|---|---|---|---|---|---|
| **Field** | Reserved | DISABLE | RELOAD | T0EN | PRESCALE |

| **Field** | **Value** | **Description** |
|---|---|---|
| Reserved | — | Reserved—write with zero. |
| DISABLE | 0 | Timer enabled. |
| | 1 | Timer and prescaler disabled. The period register and **timer0** are not reset. |
| RELOAD | 0 | Timer stops after counting down to 0. |
| | 1 | Timer automatically reloads and repeats indefinitely. |
| T0EN | 0 | Timer holds current count. |
| | 1 | Timer counts down to 0. |
| PRESCALE | — | See PRESCALE field table below. |

**PRESCALE Field**

| **PRESCALE** | **Frequency of Timer Interrupts** | **PRESCALE** | **Frequency of Timer Interrupts** |
|---|---|---|---|
| 0000 | CKO/2 | 1000 | CKO/512 |
| 0001 | CKO/4 | 1001 | CKO/1024 |
| 0010 | CKO/8 | 1010 | CKO/2048 |
| 0011 | CKO/16 | 1011 | CKO/4096 |
| 0100 | CKO/32 | 1100 | CKO/8192 |
| 0101 | CKO/64 | 1101 | CKO/16384 |
| 0110 | CKO/128 | 1110 | CKO/32768 |
| 0111 | CKO/256 | 1111 | CKO/65536 |

## 5 Software Architecture (continued)

### 5.3 Reset States

- A • indicates that this bit is unknown on powerup reset and unaffected on subsequent reset.
- An S indicates that this bit shadows the PC.
- A P indicates the value on an input pin, i.e., the bit in the register reflects the value on the corresponding input pin.
- Capital letters indicate registers that cannot be directly accessed by the instruction set.

**Table 55. Register Settings After Reset**

| Register | Bits 15—0 | Register | Bits 15—0 |
|---|---|---|---|
| r0 | •••••••••••••••• | OLIM0 | •••••••••••••••• |
| r1 | •••••••••••••••• | inc | 0000000000000000 |
| r2 | •••••••••••••••• | ins | 0000000000000010 |
| r3 | •••••••••••••••• | SSDX | •••••••••••••••• |
| j | •••••••••••••••• | saddx | •••••••••••••••• |
| k | •••••••••••••••• | cloop | 000000000••••••• |
| rb | 0000000000000000 | mwait* | 0000000000000000 |
| re | 0000000000000000 | eir | 0000000000001111 |
| pt | •••••••••••••••• | SSIOC | •••••00000000000 |
| pr | •••••••••••••••• | cbit | •••••••••••••••• |
| pi | SSSSSSSSSSSSSSSS | sbit | 00000000PPPPPPPP |
| i | •••••••••••••••• | ioc | 0000000000000000 |
| p | •••••••••••••••• | jtag | •••••••••••••••• |
| pl | •••••••••••••••• | edr | •••••••••••••••• |
| x | •••••••••••••••• | a0 | •••••••••••••••• |
| y | •••••••••••••••• | a0l | •••••••••••••••• |
| yl | •••••••••••••••• | a1 | •••••••••••••••• |
| auc | 0000000000000000 | a1l | •••••••••••••••• |
| psw | ••••00•••••••••• | timerc | ••••••••00000000 |
| c0 | •••••••••••••••• | timer0 | 0000000000000000 |
| c1 | •••••••••••••••• | ear | 0000000000000000 |
| c2 | •••••••••••••••• | powerc | 0000000000000000 |
| sioc | •••••00000000000 | pllc | 0000000000000000 |
| srta | •••••••••••••••• | ar0 | •••••••••••••••• |
| sdx | •••••••••••••••• | ar1 | •••••••••••••••• |
| tdms | ••••••0000000000 | ar2 | •••••••••••••••• |
| PHIFC | 0000000000000000 | ar3 | •••••••••••••••• |
| PDX | 0000000000000000 | alf | 00000000•••••••• |
| ybase | •••••••••••••••• | miwp1 | ••••••0000000000 |
| miwp0 | ••••••0000000000 | morp1 | ••••••0000000000 |
| morp0 | ••••••0000000000 | mcmd1 | •••••••••••••••• |
| mcmd0 | •••••••••••••••• | ILEN1 | ••••111111111111 |
| ILEN0 | ••••111111111111 | OLEN1 | •••••00000000000 |
| OLEN0 | •••••00000000000 | IBAS1 | •••••••••••••••• |
| IBAS0 | •••••••••••••••• | OBAS1 | •••••••••••••••• |
| OBAS0 | •••••••••••••••• | OLIM1 | •••••••••••••••• |

* If EXM is high and INT1 is low when RSTB goes high, **mwait** will contain all ones instead of all zeros.

## 5 Software Architecture (continued)

### 5.4 Instruction Set Formats

This section defines the hardware-level encoding of the DSP1620 device instructions.

#### Multiply/ALU Instructions

Format 1: Multiply/ALU Read/Write Group

| **Field** | T | | | | | D | S | F1 | | | | X | Y | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Format 1a: Multiply/ALU Read/Write Group

| **Field** | T | | | | | a$\overline{T}$ | S | F1 | | | | X | Y | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Format 2: Multiply/ALU Read/Write Group

| **Field** | T | | | | | D | S | F1 | | | | X | Y | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Format 2a: Multiply/ALU Read/Write Group

| **Field** | T | | | | | a$\overline{T}$ | S | F1 | | | | X | Y | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

#### Special Function Instructions

Format 3: F2 ALU Special Functions

| **Field** | T | | | | | D | S | F2 | | | | CON | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Format 3a: F3 ALU Operations

| **Field** | T | | | | | D | S | F3 | | | | SRC2 | | aT | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Immediate Operand (IM16) | | | | | | | | | | | | | | | |
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Format 3b: BMU Operations

| **Field** | T | | | | | D | S | F4[3—1] | | | 0 | F4[0] | AR | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Immediate Operand (IM16) | | | | | | | | | | | | | | | |
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

## 5 Software Architecture (continued)

### Control Instructions

Format 4: Branch Direct Group

| **Field** | T | | | | JA | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Format 5: Branch Indirect Group

| **Field** | T | | | | | B | | | Reserved | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Format 6: Conditional Branch Qualifier/Software Interrupt (**icall**)
**Note:** A branch instruction immediately follows except for a software interrupt (**icall**).

| **Field** | T | | | | | SI | Reserved | | | | | CON | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

### Data Move Instructions

Format 7: Data Move Group

| **Field** | T | | | | | $a\overline{T}$ | R | | | | | | Y/Z | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Format 8: Data Move (immediate operand—2 words) Group

| **Field** | T | | | | | D | R | | | | | | Reserved | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Immediate Operand (IM16) | | | | | | | | | | | | | | | |
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Format 9: Short Immediate Group

| **Field** | T | | | | | I | | Short Immediate Operand (IM9) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Format 9a: Direct Addressing

| **Field** | T | | | | | R/W | DR | | | | 1 | OFFSET | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

### Cache Instructions

Format 10: Do/Redo

| **Field** | T | | | | | N | | | | K | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

## 5 Software Architecture (continued)

### 5.5 Field Descriptions

**Table 56. T Field**

Specifies the type of instruction.

| T | Operation | | Format |
|---|---|---|---|
| 0000x | goto JA | | 4 |
| 00010 | Short imm j, k, rb, re | | 9 |
| 00011 | Short imm r0, r1, r2, r3 | | 9 |
| 00100 | Y = a1[l] | F1 | 1 |
| 00101 | Z : aT[l] | F1 | 2a |
| 00110 | Y | F1 | 1 |
| 00111 | aT[l] = Y | F1 | 1a |
| 01000 | Bit 0 = 0, aT = R | | 7 |
| 01000 | Bit 0 = 1, aTl = R | | 7 |
| 01001 | Bit 10 = 0, R = a0 | | 7 |
| 01001 | Bit 10 = 1, R = a0l | | 7 |
| 01010 | R = IM16 | | 8 |
| 01011 | Bit 10 = 0, R = a1 | | 7 |
| 01011 | Bit 10 = 1, R = a1l | | 7 |
| 01100 | Y = R | | 7 |
| 01101 | Z : R | | 7 |
| 01110 | do, redo | | 10 |
| 01111 | R = Y | | 7 |
| 1000x | call JA | | 4 |
| 10010 | ifc CON | F2 | 3 |
| 10011 | if CON | F2 | 3 |
| 10100 | Y = y[l] | F1 | 1 |
| 10101 | Z : y[l] | F1 | 2 |
| 10110 | x = Y | F1 | 1 |
| 10111 | y[l] = Y | F1 | 1 |
| 11000 | Bit 0 = 0, branch indirect | | 5 |
| 11000 | Bit 0 = 1, F3 ALU | | 3a |
| 11001 | y = a0 x = X | F1 | 1 |
| 11010 | Cond. branch qualifier | | 6 |
| 11011 | y = a1 x = X | F1 | 1 |
| 11100 | Y = a0[l] | F1 | 1 |
| 11101 | Z : y x = X | F1 | 2 |
| 11110 | Bit 5 = 0, F4 ALU (BMU) | | 3b |
| 11110 | Bit 5 = 1, direct addressing | | 9a |
| 11111 | y = Y x = X | F1 | 1 |

**Table 57. D Field**

Specifies a destination accumulator.

| D | Register |
|---|---|
| 0 | Accumulator 0 |
| 1 | Accumulator 1 |

**Table 58. a$\overline{T}$ Field**

Specifies a transfer accumulator.

| aT | a$\overline{T}$ | Register |
|---|---|---|
| 1 | 0 | Accumulator 1 |
| 0 | 1 | Accumulator 0 |

**Table 59. S Field**

Specifies a source accumulator.

| S | Register |
|---|---|
| 0 | Accumulator 0 |
| 1 | Accumulator 1 |

**Table 60. F1 Field**

Specifies the multiply/ALU function.

| F1 | Operation | |
|---|---|---|
| 0000 | aD = p | p = x * y |
| 0001 | aD = aS + p | p = x * y |
| 0010 | | p = x * y |
| 0011 | aD = aS – p | p = x * y |
| 0100 | aD = p | |
| 0101 | aD = aS + p | |
| 0110 | nop | |
| 0111 | aD = aS – p | |
| 1000 | aD = aS \| y | |
| 1001 | aD = aS ^ y | |
| 1010 | aS & y | |
| 1011 | aS – y | |
| 1100 | aD = y | |
| 1101 | aD = aS + y | |
| 1110 | aD = aS & y | |
| 1111 | aD = aS – y | |

**Table 61. X Field**

Specifies the addressing of ROM data in two-operand multiply/ALU instructions. Specifies the high or low half of an accumulator or the **y** register in one-operand multiply/ALU instructions.

| X | Operation |
|---|---|
| **Two-Operand Multiply/ALU** | |
| 0 | *pt++ |
| 1 | *pt++i |
| **One-Operand Multiply/ALU** | |
| 0 | aTl, yl |
| 1 | aTh, yh |

## 5 Software Architecture (continued)

**Table 62. Y Field**

Specifies the form of register indirect addressing with postmodification.

| Y | Operation |
|---|---|
| 0000 | *r0 |
| 0001 | *r0++ |
| 0010 | *r0-- |
| 0011 | *r0++j |
| 0100 | *r1 |
| 0101 | *r1++ |
| 0110 | *r1-- |
| 0111 | *r1++j |
| 1000 | *r2 |
| 1001 | *r2++ |
| 1010 | *r2-- |
| 1011 | *r2++j |
| 1100 | *r3 |
| 1101 | *r3++ |
| 1110 | *r3-- |
| 1111 | *r3++j |

**Table 63. Z Field**

Specifies the form of register indirect compound addressing with postmodification.

| Z | Operation |
|---|---|
| 0000 | *r0zp |
| 0001 | *r0pz |
| 0010 | *r0m2 |
| 0011 | *r0jk |
| 0100 | *r1zp |
| 0101 | *r1pz |
| 0110 | *r1m2 |
| 0111 | *r1jk |
| 1000 | *r2zp |
| 1001 | *r2pz |
| 1010 | *r2m2 |
| 1011 | *r2jk |
| 1100 | *r3zp |
| 1101 | *r3pz |
| 1110 | *r3m2 |
| 1111 | *r3jk |

**Table 64. F2 Field**

Specifies the special function to be performed.

| F2 | Operation |
|---|---|
| 0000 | aD = aS >> 1 |
| 0001 | aD = aS << 1 |
| 0010 | aD = aS >> 4 |
| 0011 | aD = aS << 4 |
| 0100 | aD = aS >> 8 |
| 0101 | aD = aS << 8 |
| 0110 | aD = aS >> 16 |
| 0111 | aD = aS << 16 |
| 1000 | aD = p |
| 1001 | aDh = aSh + 1 |
| 1010 | aD = ~aS |
| 1011 | aD = rnd(aS) |
| 1100 | aD = y |
| 1101 | aD = aS + 1 |
| 1110 | aD = aS |
| 1111 | aD = – aS |

**Table 65. CON Field**

Specifies the condition for special functions and conditional control instructions.

| CON | Condition | CON | Condition |
|---|---|---|---|
| 00000 | mi | 10000 | gt |
| 00001 | pl | 10001 | le |
| 00010 | eq | 10010 | allt |
| 00011 | ne | 10011 | allf |
| 00100 | lvs | 10100 | somet |
| 00101 | lvc | 10101 | somef |
| 00110 | mvs | 10110 | oddp |
| 00111 | mvc | 10111 | evenp |
| 01000 | heads | 11000 | mns1 |
| 01001 | tails | 11001 | nmns1 |
| 01010 | c0ge | 11010 | npint |
| 01011 | c0lt | 11011 | njint |
| 01100 | c1ge | 11100 | lock |
| 01101 | c1lt | 11101 | ebusy |
| 01110 | true | 11110 | mioubusy |
| 01111 | false | Other codes | Reserved |

## 5 Software Architecture (continued)

**Table 66. R Field**

Specifies the register for data move instructions.

| R | Register | R | Register |
|---|---|---|---|
| 000000 | r0 | 100000 | inc |
| 000001 | r1 | 100001 | ins |
| 000010 | r2 | 100010 | miwp0 |
| 000011 | r3 | 100011 | saddx |
| 000100 | j | 100100 | cloop |
| 000101 | k | 100101 | mwait |
| 000110 | rb | 100110 | morp0 |
| 000111 | re | 100111 | mcmd0 |
| 001000 | pt | 101000 | cbit |
| 001001 | pr | 101001 | sbit |
| 001010 | pi | 101010 | ioc |
| 001011 | i | 101011 | jtag |
| 001100 | p | 101100 | Reserved |
| 001101 | pl | 101101 | Reserved |
| 001110 | pllc | 101110 | Reserved |
| 001111 | Reserved | 101111 | eir |
| 010000 | x | 110000 | a0 |
| 010001 | y | 110001 | a0l |
| 010010 | yl | 110010 | a1 |
| 010011 | auc | 110011 | a1l |
| 010100 | psw | 110100 | timerc |
| 010101 | c0 | 110101 | timer0 |
| 010110 | c1 | 110110 | mcmd1 |
| 010111 | c2 | 110111 | Reserved |
| 011000 | sioc | 111000 | powerc |
| 011001 | srta | 111001 | edr |
| 011010 | sdx | 111010 | ar0 |
| 011011 | tdms | 111011 | ar1 |
| 011100 | miwp1 | 111100 | ar2 |
| 011101 | morp1 | 111101 | ar3 |
| 011110 | Reserved | 111110 | ear |
| 011111 | ybase | 111111 | alf |

**Table 67. B Field**

Specifies the type of branch instruction (except software interrupt).

| B | Operation |
|---|---|
| 000 | return |
| 001 | ireturn |
| 010 | goto pt |
| 011 | call pt |
| 1xx | Reserved |

**Table 68. DR Field**

| DR | Register |
|---|---|
| 0000 | r0 |
| 0001 | r1 |
| 0010 | r2 |
| 0011 | r3 |
| 0100 | a0 |
| 0101 | a0l |
| 0110 | a1 |
| 0111 | a1l |
| 1000 | y |
| 1001 | yl |
| 1010 | p |
| 1011 | pl |
| 1100 | x |
| 1101 | pt |
| 1110 | pr |
| 1111 | psw |

**Table 69. I Field**

Specifies a register for short immediate data move instructions.

| I | Register |
|---|---|
| 00 | r0/j |
| 01 | r1/k |
| 10 | r2/rb |
| 11 | r3/re |

**Table 70. SI Field**

Specifies when the conditional branch qualifier instruction should be interpreted as a software interrupt instruction.

| SI | Operation |
|---|---|
| 0 | Not a software interrupt |
| 1 | Software interrupt |

## 5 Software Architecture (continued)

### N Field

Number of instructions to be loaded into the cache. Zero implies redo operation.

### K Field

Number of times the N instructions in cache are to be executed. Zero specifies use of value in **cloop** register.

### JA Field

12-bit jump address.

### R/W Field

A zero specifies a write, ***(OFFSET) = DR**.
A one specifies a read, **DR = *(OFFSET)**

**Table 71. F3 Field**

Specifies the operation in an F3 ALU instruction.

| F3 | Operation | | |
|---|---|---|---|
| 1000 | aD = aS[h, l] | \| | {aT, IM16, p} |
| 1001 | aD = aS[h, l] | ^ | {aT, IM16, p} |
| 1010 | aS[h, l] | & | {aT, IM16, p} |
| 1011 | aS[h, l] | – | {aT, IM16, p} |
| 1101 | aD = aS[h, l] | + | {aT, IM16, p} |
| 1110 | aD = aS[h, l] | & | {aT, IM16, p} |
| 1111 | aD = aS[h, l] | – | {aT, IM16, p} |

**Table 72. SRC2 Field**

Specifies operands in an F3 ALU instruction.

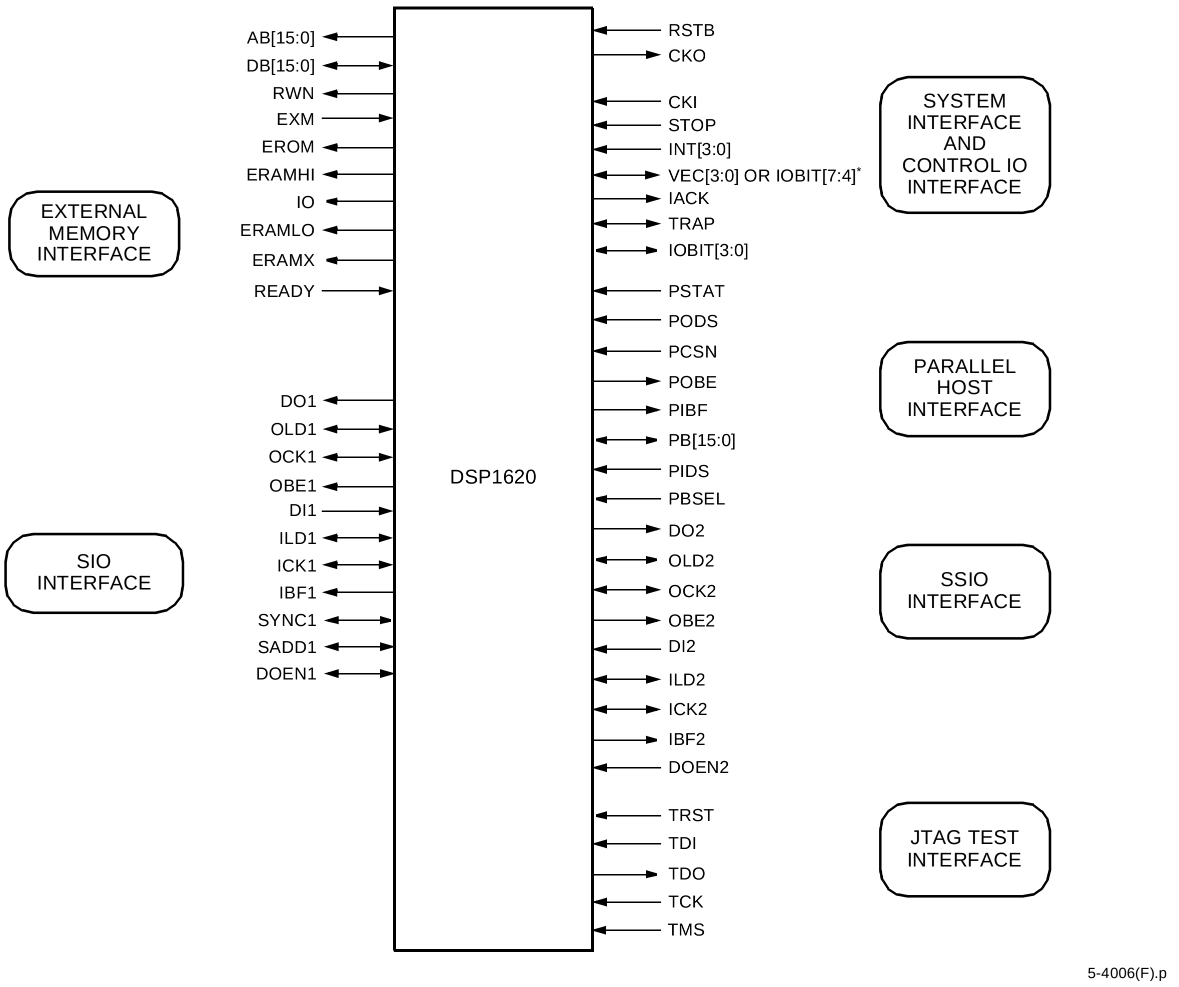
| SRC2 | Operands |
|---|---|
| 00 | aSl, IM16 |
| 01 | aS, aT |
| 10 | aSh, IM16 |
| 11 | aS, p |

**Table 73. F4 and AR Fields**

| F4 | AR | Operation |
|---|---|---|
| 0000 | 00xx | aD = aS >> arM |
| 0001 | 00xx | aD = aS << arM |
| 0000 | 10xx | aD = aS >>> arM |
| 0001 | 10xx | aD = aS <<< arM |
| 1000 | 0000 | aD = $\overline{\text{aS}}$ >> aS |
| 1001 | 0000 | aD = $\overline{\text{aS}}$ << aS |
| 1000 | 1000 | aD = $\overline{\text{aS}}$ >>> aS |
| 1001 | 1000 | aD = $\overline{\text{aS}}$ <<< aS |
| 1100 | 0000 | aD = aS >> IM16 |
| 1101 | 0000 | aD = aS << IM16 |
| 1100 | 1000 | aD = aS >>> IM16 |
| 1101 | 1000 | aD = aS <<< IM16 |
| 0000 | 1100 | aD = exp(aS) |
| 0001 | 11xx | aD = norm(aS, arM) |
| 1110 | 0000 | aD = extracts(aS, IM16) |
| 0010 | 00xx | aD = extracts(aS, arM) |
| 1110 | 0100 | aD = extractz(aS, IM16) |
| 0010 | 01xx | aD = extractz(aS, arM) |
| 1110 | 1000 | aD = insert(aS, IM16) |
| 1010 | 10xx | aD = insert(aS, arM) |
| 0111 | 0000 | aD = aS:aa0 |
| 0111 | 0001 | aD = aS:aa1 |

**Note:** xx encodes the auxiliary register to be used. 00 (**ar0**), 01(**ar1**), 10 (**ar2**), or 11(**ar3**).

## 6 Signal Descriptions



* Note that VEC0 corresponds to IOBIT7, VEC1 corresponds to IOBIT6, VEC2 corresponds to IOBIT5, and VEC3 corresponds to IOBIT4.

**Figure 15. DSP1620 Pinout by Interface**

Figure 15 shows the pinout for the DSP1620. The signals can be separated into six interfaces as shown. These interfaces and the signals that comprise them are described below.

## 6 Signal Descriptions (continued)

### 6.1 System Interface

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

#### RSTB

**Reset:** Negative assertion. A high-to-low transition causes the processor to enter the reset state. The **auc**, **powerc**, **sioc**, **SSIOC**, **PHIFC**, **PDX**, **tdms**, **timerc**, **timer0**, **sbit** (upper byte), **inc**, **ins** (except OBE is set), **alf** (upper 2 bits, AWAIT and LOWPR), **ioc**, **rb**, **re**, **ILEN0**, **OLEN0**, **ILEN1**, **OLEN1**, **miwp0**, **miwp1**, **morp0**, and **morp1** are initialized to known values.

The **mwait** register is initialized to all 0s (zero wait-states) unless the EXM pin is high and the INT1 pin is low. In that case, the **mwait** register is initialized to all 1s (15 wait-states).

Reset clears IACK, VEC[3:0]/IOBIT[4:7], IBF, PIBF, and IBF2. The DAU condition flags are not affected by reset. IOBIT[7:0] are initialized as inputs. If any of the IOBIT pins are switched to outputs (by writing **sbit**), their initial value will be logic zero (see Figure 55).

Upon negation of the signal, the processor begins execution at location 0x0000 in the active memory map (see Section 4.4, Memory Maps and Wait-States).

#### CKI

**Input Clock:** The CKI input buffer drives the internal processor clock directly (1X) or drives the on-chip PLL (see Section 4.16). The PLL allows the CKI input clock to be at a lower frequency than the internal processor clock.

#### STOP

**Stop Input Clock:** Negative assertion. A high-to-low transition synchronously stops all of the internal processor clocks leaving the processor in a defined state. Returning the pin high will synchronously restart the processor clocks 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 45).

#### CKO

**Clock Out:** Buffered output clock with options programmable via the **ioc** register (see Table 41). The selectable CKO options are as follows:

- A free-running output clock at the frequency of the internal processor clock; runs at the internal ring oscillator frequency when SLOWCKI is enabled.
- A wait-stated clock based on the internal instruction cycle; runs at the internal ring oscillator frequency when SLOWCKI is enabled.
- A sequenced, wait-stated clock based on the EMI sequencer cycle; runs at the internal ring oscillator frequency when SLOWCKI is enabled.
- A free-running output clock that runs at the CKI rate, independent of the **powerc** register setting. When the PLL is selected, the CKO frequency equals the input CKI frequency regardless of how the PLL is programmed.
- A logic 0.
- A logic 1.

#### INT[3:0]

**Processor Interrupts 3, 2, 1, and 0:** Positive assertion. Hardware interrupt inputs to the DSP1620. Each is enabled via the **inc** register. When enabled and asserted, each cause the processor to vector to the memory location described in Table 4. INT1 is used in conjunction with EXM to select the desired reset initialization of the **mwait** register (see Table 42). **When the level of RSTB = 0 and INT0 = 1, all output and bidirectional pins (except TDO, which 3-states by JTAG control) are put in a 3-state condition (refer to Table 1 and its related footnote).**

#### VEC[3:0]

**Interrupt Output Vector:** These four pins indicate which interrupt is currently being serviced by the device. Table 4 shows the code associated with each interrupt condition. VEC[3:0] are multiplexed with IOBIT[4:7].

#### IACK

**Interrupt Acknowledge:** Positive assertion. IACK signals when an interrupt is being serviced by the DSP1620. IACK remains asserted while in an interrupt service routine, and is cleared when the **ireturn** instruction is executed.

#### TRAP

**Trap Signal:** Positive assertion. When asserted, the processor is put into the trap condition, which normally causes a branch to the location 0x0046. The hardware development system (HDS) can configure the trap pin to cause an HDS trap, which causes a branch to location 0x0003. Although normally an input, the pin can be configured as an output by the HDS. As an output, the pin can be used to signal an HDS breakpoint in a multiple processor environment.

## 6 Signal Descriptions (continued)

### 6.2 External Memory Interface

The external memory interface is used to interface the DSP1620 to external memory and I/O devices. It supports read/write operations from/to program and data memory spaces. The interface supports four external memory segments. Each external memory segment can have an independent number of software programmable wait-states.

#### AB[15:0]

**External Memory Address Bus:** Output only. This 16-bit bus supplies the address for read or write operations to the external memory or I/O. During internal memory accesses, AB[15:0] retain the value of the last valid external access.

#### DB[15:0]

**External Memory Data Bus:** This 16-bit bidirectional data bus is used for read or write operations to the external memory or I/O.

#### RWN

**Read/Write Not:** When a logic 1, the pin indicates that the data memory access (Y) is a read operation. When a logic 0, the memory access is a write operation.

#### EXM

**External Memory Select:** Input only. This signal is latched into the device on the rising edge of RSTB. The value of EXM latched in determines whether the internal ROM is addressable in the instruction/coefficient memory map. If EXM is low, internal ROM is addressable. If EXM is high, only external ROM (EROM) is addressable in the instruction/coefficient memory map (see Table 5, Instruction/Coefficient Memory Maps). EXM chooses between XMAP1 or XMAP2.

#### EROM

**External ROM Enable Signal:** Negative assertion. When asserted, the signal indicates an access (either X or Y) to the EROM segment (see Table 5 and Table 6). This signal's leading edge can be delayed via the **ioc** register (see Table 41).

#### ERAMHI

**External RAM High Enable Signal:** Negative assertion. When asserted, the signal indicates a Y access to the external ERAMHI segment (see Table 6). This signal's leading edge can be delayed via the **ioc** register (see Table 41).

#### ERAMLO

**External RAM Low Enable Signal:** Negative assertion. When asserted, the signal indicates a Y access to the external ERAMLO segment (see Table 6). This signal's leading edge can be delayed via the **ioc** register (see Table 41).

#### IO

**External I/O Enable Signal:** Negative assertion. When asserted, the signal indicates an access to external data memory-mapped I/O segment (see Table 6). This signal's leading edge can be delayed via the **ioc** register (see Table 41).

#### ERAMX

**External RAM Enable Signal:** Negative assertion. When asserted, the signal indicates an access to either the ERAMHI, ERAMLO, or I/O external memory segments (see Table 41). This signal's leading edge can be delayed via the **ioc** register (see Table 41).

#### READY

**External Ready Acknowledge:** Positive assertion level. When set low (0), DSP instruction execution is stalled until the signal is set high.

### 6.3 SIO Interface

The SIO interface pins implement a full-featured synchronous/asynchronous serial I/O channel. In addition, several pins offer a glueless TDM interface for multiprocessing communication applications (see Figure 8, Multiprocessor Communications and Connections).

#### DI1

**Data Input:** Serial data is latched on the rising edge of ICK1, either LSB or MSB first, according to the **sioc** register MSB field (see Table 50).

## 6 Signal Descriptions (continued)

**ICK1**

**Input Clock:** The clock for serial input data. In active mode, ICK1 is an output; in passive mode, ICK1 is an input, according to the **sioc** register ICK field (see Table 50). Input has typically 0.7 V hysteresis.

**ILD1**

**Input Load:** The clock for loading the input buffer, **sdx**(in), from the input shift register **isr**. A falling edge of ILD1 indicates the beginning of a serial input word. In active mode, ILD1 is an output; in passive mode, ILD1 is an input, according to the **sioc** register ILD field (see Table 50). Input has typically 0.7 V hysteresis.

**IBF1**

**Input Buffer Full:** Positive assertion. IBF1 is asserted when the input buffer, **sdx**(in), is filled. IBF1 is negated by a read of the buffer, as in **a0** = **sdx**. IBF1 is also negated by asserting RSTB.

**DO1**

**Data Output:** The serial data output from the output shift register (**osr**), either LSB or MSB first (according to the **sioc** register MSB field). DO1 normally changes on the rising edges of OCK1 but can be programmed to change on falling edges, as determined by the DODLY field of the **sioc** register. DO1 is 3-stated when DOEN1 is high.

**DOEN1**

**Data Output Enable:** Negative assertion. An input when not in the multiprocessor mode. DO1 and SADD1 are enabled only if DOEN1 is low. DOEN1 is bidirectional when in the multiprocessor mode (**tdms** register MODE field set). In the multiprocessor mode, DOEN1 indicates a valid time slot for a serial output.

**OCK1**

**Output Clock:** The clock for serial output data. In active mode, OCK1 is an output; in passive mode, OCK1 is an input, according to the **sioc** register OCK field (see Table 50). Input has typically 0.7 V hysteresis.

**OLD1**

**Output Load:** The clock for loading the output shift register, **osr**, from the output buffer **sdx**(out). A falling edge of OLD1 indicates the beginning of a serial output word. In active mode, OLD1 is an output; in passive, OLD1 is an input, according to the **sioc** register OLD field (see Table 50). Input has typically 0.7 V hysteresis.

**OBE1**

**Output Buffer Empty:** Positive assertion. OBE1 is asserted when the output buffer, **sdx**(out), is emptied (moved to the output shift register for transmission). It is cleared with a write to the buffer, as in **sdx** = **a0**. OBE1 is also set by asserting RSTB.

**SADD1**

**Serial Address:** Negative assertion. A 16-bit serial bit stream typically used for addressing during multiprocessor communication between multiple DSP16xx devices. In multiprocessor mode, SADD1 is an output when the **tdms** time slot dictates a serial transmission; otherwise, it is an input. Both the source and destination DSP can be identified in the transmission. SADD1 is always an output when not in multiprocessor mode and can be used as a second 16-bit serial output. See the *DSP1620 Digital Signal Processor* Information Manual for additional information. SADD1 is 3-stated when DOEN1 is high. When used on a bus, SADD1 should be pulled high through a 5 kΩ resistor.

**SYNC1**

**Multiprocessor Synchronization:** Typically used in the multiprocessor mode, a falling edge of SYNC1 indicates the first word of a TDM I/O stream and causes resynchronization of the active ILD1 and OLD1 generators. SYNC1 is an output if the **tdms** register SYNC field is set (i.e., selects the master DSP and uses time slot 0 for transmit). As an input, SYNC1 must be tied low unless part of a TDM interface. If used as an output, SYNC1 = [ILD1/OLD1]/8 or 16 according on the setting of the SYNCSP field of the **tdms** register. If configured as described above, SYNC1 can be used to generate a slow clock for SIO operations. Input has typically 0.7 V hysteresis.

### 6.4 SSIO Interface

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

**DI2**

**Data Input:** Serial data is latched on the rising edge of ICK2, either LSB or MSB first, according to the **SSIOC** register MSB2 field (see Table 52).

**ICK2**

**Input Clock:** The clock for serial input data. In active mode, ICK2 is an output; in passive mode, ICK2 is an input, according to the **SSIOC** register ICK2 field (see Table 52). Input has typically 0.7 V hysteresis.

## 6 Signal Descriptions (continued)

### ILD2

**Input Load:** The clock for loading the input buffer, **SSDX**(in), from the input shift register **isr**. A falling edge of ILD2 indicates the beginning of a serial input word. In active mode, ILD2 is an output; in passive mode, ILD2 is an input, according to the **SSIOC** register ILD field (see Table 52). Input has typically 0.7 V hysteresis.

### IBF2

**Input Buffer Full:** Positive assertion. IBF2 is asserted when the input buffer, **SSDX**(in), is filled. IBF2 is reset when MIOU0 transfers **SSDX**(in) to IORAM bank 32. IBF2 is also reset by asserting RSTB.

### DO2

**Data Output:** The serial data output from the output shift register (**osr**), either LSB or MSB first (according to the **SSIOC** register MSB2 field). DO2 normally changes on the rising edges of OCK2 but can be programmed to change on falling edges, as determined by the DODLY2 field of the **SSIOC** register. DO2 is 3-stated when DOEN2 is high.

### DOEN2

**Data Output Enable:** Negative assertion. DO2 is enabled only if DOEN2 is low.

### OCK2

**Output Clock:** The clock for serial output data. In active mode, OCK2 is an output; in passive mode, OCK2 is an input, according to the **SSIOC** register OCK2 field (see Table 52). Input has typically 0.7 V hysteresis.

### OLD2

**Output Load:** The clock for loading the output shift register, **osr**, from the output buffer **SSDX**(out). A falling edge of OLD2 indicates the beginning of a serial output word. In active mode, OLD2 is an output; in passive, OLD2 is an input, according to the **SSIOC** register OLD2 field (see Table 52). Input has typically 0.7 V hysteresis.

### OBE2

**Output Buffer Empty:** Positive assertion. OBE2 is asserted when the output buffer, **SSDX**(out), is emptied (moved to the output shift register for transmission). OBE2 is cleared when MIOU0 fills **SSDX**(out).

## 6.5 PHIF16 Interface

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

### PB[15:0]

**Parallel I/O Data Bus:** 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. When configured as an 8-bit bus, PB[15:8] are 3-stated.

### PCSN

**Peripheral Chip Select Not:** Negative assertion. PCSN is an input. While PCSN is low, the data strobes PIDS and PODS are enabled. While PCSN is high, the DSP1620 ignores any activity on PIDS and PODS.

### PBSEL

**Peripheral Byte Select:** An input pin, configurable in software. Selects the high or low byte of **PDX** available for host accesses (8-bit external mode).

### PSTAT

**Peripheral Status Select:** PSTAT is an input. When a logic 0, the PHIF16 will output the **PDX**(out) register on the PB bus. When a logic 1, the PHIF16 will output the contents of the PSTAT register on PB[7:0].

### PIDS

**Parallel Input Data Strobe:** An input pin, software configurable to support both *Intel* and *Motorola* protocols.

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.

## 6 Signal Descriptions (continued)

**PODS**

**Parallel Output Data Strobe:** An input pin, software configurable to support both *Intel* and *Motorola* protocols.

In *Intel* mode: Negative assertion. When PODS is pulled low by an external device, the DSP1620 places the contents of the parallel output register, **PDX**, 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**

**Parallel Input Buffer Full:** An output pin with positive assertion; configurable in software. This flag is cleared after reset, indicating an empty input buffer **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 DSP1620 reads the contents of this register, emptying the buffer, the flag is cleared.

Configured in software, PIBF can become the logical OR of the PIBF and POBE flags.

**POBE**

**Parallel Output Buffer Empty:** An output pin with positive assertion; configurable in software. This flag is set after reset, indicating an empty output buffer **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 DSP1620 writes to **PDX**(out), filling the buffer, this flag is cleared.

### 6.6 Control I/O Interface

This interface is used for status and control operations provided by the bit I/O unit of the DSP1620. It is pin multiplexed with the VEC[3:0] pins (see Section 4.1). Setting the EBIOH bit in the **ioc** register provides a full 8-bit BIO interface at the associated pins.

**IOBIT[7:0]**

**I/O Bits [7:0]:** Each of these bits 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.

### 6.7 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 DSP1620 provides all JTAG/*IEEE* 1149.1 standard test capabilities including boundary-scan. See the *DSP1620 Digital Signal Processor* Information Manual for additional information on the JTAG test interface.

**TDI**

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

**TDO**

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

**TMS**

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

**TCK**

**Test Clock:** JTAG 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**

**Test Reset:** Negative assertion. JTAG test reset. When asserted low, asynchronously resets JTAG TAP controller. In an application environment, this pin must be asserted prior to or concurrent with RSTB. This pin has an internal pull-up resistor.

# 7 DSP1620 Boot Routines

There are more than 50 subroutines in the internal ROM of the DSP1620 that allow the user to perform various functions immediately after powerup of the device. The primary function of these routines is to allow the user to download code and data to the internal DPRAM of the DSP1620 from the PHIF16, SIO, or EMI ports. Other routines include download to external ROM (EROM) space, memory test routines (both internal and external), and a routine that enables or disables the PLL. Once the user has finished the download of code and/or data to a selected memory segment, the user can select the appropriate XMAP for execution.

## 7.1 Logical Flow of Events

When the 1620 is powered up with the EXM pin low, XMAP1 is selected and the device begins the execution of code starting at address 0x0 in internal ROM. Each boot routine begins execution by configuring the host interface port, PHIF16, for a single, 8-bit *Intel* style transfer and enabling INT3. The PLL is left disabled at this time. This is done so a single strobe executes the transfer to interface to a *Motorola* compatible interface (some external logic may be required). DSP1620 then waits for a host device interfaced to the PHIF16 to write an 8-bit value onto PB[7:0] and strobe PIDS. Once the DSP1620 has input this value, it is decoded to select the appropriate boot routine and set the selected number of wait-states (if external memory is involved). INT3 is used to terminate any download routine under host control; otherwise, the routine will self-terminate when the entire memory segment is filled. At the completion of the download routine, a value of 0xED is written out the PHIF16 port to the host to indicate that the download is complete and the DSP core has completed the transfer of data to the selected memory space. In response to the POBE flag, the host must read this value from the port in order for the code to continue. The program flow is stalled until the port is read. After the handshake byte has been read, the PHIF port is returned to the 8-bit *Intel* style mode and waits for the next command.

If a 0x0 or invalid command is detected, the entry is ignored and the program looks for the next command from the host.

Once the user has downloaded code and data to the selected memory segments, the host can direct DSP1620 to begin execution from any XMAP by selecting the appropriate boot routine to switch to that map. DSP1620 can then switch to XMAP1, 3, or 4 completely under software control without further host intervention. Execution from XMAP2 requires the host to set EXM = 1. In the event that this map is desired, the DSP1620 waits for the host to set EXM high and issues a reset by toggling RSTB (which latches EXM). After RSTB goes high with EXM = 1, execution begins from address 0x0 in external ROM (EROM).

With the exception of the XMAP routines, all boot routines, upon completion, place the device into handshake byte mode, where a 0xED is output from the PHIF16 port to the host. The host must read the port in order for the code to continue. After the handshake byte has been read, the code awaits the next command from the host. All commands from the host must be issued as 8-bit *Intel* style transfers. The format of the commands are:

PB[7:6] select wait-states if external memory is used, the active clock divider for SIO download routines (**sioc** register, CLK1 field), the PHIF16 configuration (mode) for the PHIF16 output routine, or CKI multiplier for the PLL routines. Options are as follows:

- 00 selects zero wait-states for external memory accesses; CKI/2 active clock for SIO; *Motorola* mode for PHIF16 output; disable the PLL and run from 1X CKI.
- 01 selects one wait-state for external memory accesses; CKI/6 active clock for SIO; *Motorola* mode with active-low PDS for PHIF16 output; enable the PLL with a 2X CKI multiplier.
- 10 selects four wait-states for external memory accesses; CKI/8 active clock for SIO; *Intel* mode for PHIF16 output; enable the PLL with a 5X CKI multiplier.
- 11 selects 15 wait-states for external memory accesses; CKI/10 active clock for SIO; *Motorola* mode for PHIF16 output; enables the PLL with a 10X CKI multiplier.

PB[5:0] select the boot routine to be executed. These are discussed below. Valid options:

- 0x0000—No action. Return to top of code and await next command.

Some registers are not returned to their reset state following execution of the boot routine utilities in the following cases:

- The boot routines have options to download code/data from the PHIF16 port to DPRAM and EROM — "Download of DPRAM from PHIF16" and "Download of EROM from PHIF16". These boot routines use virtual shift addressing (circular buffering). The **re** register is not cleared to 0 prior to exiting the boot routine to disable the circular buffers. **Therefore, the DSP programmer MUST set re = 0 in their application code to disable circular buffering if PHIF16 boot download routines were executed prior to the application code's execution.**

## 7 DSP1620 Boot Routines (continued)

- The **inc** register is not reset to 0x0000 following completion of the boot routine. The **inc** register is used to enable **int3**. Asserting **int3** during boot code operation terminates the current routine and resets the boot code to wait for the next command.
- The Modular I/O Units (MIOU) and their associate peripherals are not reset after completion of the boot routine uploads. Before using the MIOU peripherals, the application code must reset the appropriate MIOU, initializing all pointers. The **OLIM**, **OBAS**, and **IBAS** registers must be programmed to known values. The **PHIFC** and **SSIOC** peripheral control registers must also be programmed to known values.

### Download to Internal DPRAM

PB[5:0]

0x0001 - Download of DPRAM from PHIF16, 8-bit external, 8-bit logical, *Motorola* mode.
0x0002 - Download of DPRAM from PHIF16, 8-bit external, 8-bit logical, *Motorola* mode, PDS active-high.
0x0003 - Download of DPRAM from PHIF16, 8-bit external, 8-bit logical, *Intel* mode.
0x0006 - Download of DPRAM from PHIF16, 8-bit external, 16-bit logical, *Motorola* mode.
0x0007 - Download of DPRAM from PHIF16, 8-bit external, 16-bit logical, *Motorola* mode, PDS active-high.
0x0008 - Download of DPRAM from PHIF16, 8-bit external, 16-bit logical, *Intel* mode.
0x0009 - Download of DPRAM from PHIF16, 16-bit external, *Motorola* mode, non-DMA.
0x000a - Download of DPRAM from PHIF16, 16-bit external, *Motorola* mode, PDS active-high, non-DMA.
0x000b - Download of DPRAM from PHIF16, 16-bit external, *Intel* mode, non-DMA.
0x000c - Download of DPRAM from PHIF16, 16-bit external, *Motorola* mode, DMA transfers.
0x000d - Download of DPRAM from PHIF16, 16-bit external, *Motorola* mode, PDS active-high, DMA transfers.
0x000e - Download of DPRAM from PHIF16, 16-bit external, *Intel* mode, DMA transfers.
0x0010 - Download of DPRAM from SIO, 8-bit, active clocks and loads.
0x0011 - Download of DPRAM from SIO, 8-bit, passive clocks and loads.
0x0012 - Download of DPRAM from SIO, 16-bit, active clocks and loads.
0x0013 - Download of DPRAM from SIO, 16-bit, passive clocks and loads.
0x0016 - Download of DPRAM from PHIF16, 16-bit external, *Motorola* mode, PDS active-low no DMA, bytes swapped.
0x0017 - Download of DPRAM from PHIF16, 16-bit external, *Motorola* mode, PDS active-high no DMA, bytes swapped.
0x0018 - Download of DPRAM from consecutive reads of I/O address 0x8000, 8-bit.
0x0019 - Download of DPRAM from consecutive reads of I/O address 0x8000, 16-bit.
0x001c - Download of DPRAM from PHIF16, 16-bit external, *Motorola* mode, PDS active-low 64 word to DMA, bytes swapped.
0x001d - Download of DPRAM from PHIF16, 16-bit external, *Motorola* mode, PDS active-high 64 word to DMA, bytes swapped.

### PLL Enable

PB[5:0]

0x001e - Enable/disable the PLL, 3 V operation.
0x001f - Enable/disable the PLL, 5 V operation.

### Download to External ROM, 32 Kwords Starting at 0x8000

PB[5:0]

0x0004 - Download of EROM from PHIF16, 16-bit external, *Motorola* mode, PDS active-low, no DMA, bytes swapped.
0x0005 - Download of EROM from PHIF16, 16-bit external, *Motorola* mode, PDS active-high, no DMA, bytes swapped.
0x0020 - Download of EROM from PHIF16, 8-bit external, 8-bit logical, *Motorola* mode.
0x0021 - Download of EROM from PHIF16, 8-bit external, 8-bit logical, *Motorola* mode, PDS active-high.
0x0022 - Download of EROM from PHIF16, 8-bit external, 8-bit logical, *Intel* mode.
0x0023 - Download of EROM from PHIF16, 8-bit external, 16-bit logical, *Motorola* mode.
0x0024 - Download of EROM from PHIF16, 8-bit external, 16-bit logical, *Motorola* mode, PDS active-high.
0x0025 - Download of EROM from PHIF16, 8-bit external, 16-bit logical, *Intel* mode.
0x0026 - Download of EROM from PHIF16, 16-bit external, *Motorola* mode, non-DMA.
0x0027 - Download of EROM from PHIF16, 16-bit external, *Motorola* mode, PDS active-high, non-DMA.
0x0028 - Download of EROM from PHIF16, 16-bit external, *Intel* mode, non-DMA.
0x000f - Download of EROM from PHIF16, 16-bit external, *Motorola* mode, PDS active-low 64 word DMA, bytes swapped.
0x002f - Download of EROM from PHIF16, 16-bit external, *Motorola* mode, PDS active-high 64 word DMA, bytes swapped.

### Download to External ROM, 64 Kwords Starting at 0x8000

PB[5:0]

0x0029 - Download of EROM from PHIF16, 16-bit external, *Motorola* mode, DMA transfers.
0x002a - Download of EROM from PHIF16, 16-bit external, *Motorola* mode, PDS active-high, DMA transfers.
0x002b - Download of EROM from PHIF16, 16-bit external, *Intel* mode, DMA transfers.

## 7 DSP1620 Boot Routines (continued)

### Memory Checksum

PB[5:0]

0x002c - Checksum of internal DPRAM, in whole (30 Kwords) or as 2 Kword blocks.
0x002d - Checksum of EROM, YMAP3.
0x002e - Checksum of EROM, YMAP4.

### Memory Test

PB[5:0]

0x0030 - Internal DPRAM test, 30 Kwords.
0x0031 - ERAMLO test, YMAP1.
0x0032 - ERAMLO test, YMAP2.
0x0033 - ERAMHI test, YMAP1.
0x0034 - ERAMHI test, YMAP2.
0x0035 - ERAMX test, YMAP1.
0x0037 - ERAMX test, YMAP2.
0x0038 - EROM test, YMAP3.
0x0039 - EROM test, YMAP4.
0x003a - RESERVED for production, RAM test.
0x003b - RESERVED for production, ROM test.

### XMAP Select and Execute

PB[5:0]

0x003c - Branch to 0x8000 (EROM) and begin execution.
0x003d - Force XMAP1 and go to sleep. Wait for host to set EXM = 1 and issue RSTB for XMAP2 execution (EROM).
0x003e - Force XMAP3 and begin execution from address 0x0 in DPRAM.
0x003f - Force XMAP4 and begin execution from address 0x0 in DPRAM.

### PHIF16 Output

PB[5:0]

0x0015 - Read results of previously run memory test.

## 7 DSP1620 Boot Routines (continued)

The DSP performs all data and command transfers in a polled mode. In the cases where wait-states are requested, all memory sections are assigned the selected number of wait-states.

### 7.2 Description of Routines

All DPRAM download routines fill 30 Kwords of DPRAM (not IORAM) unless terminated by INT3. All non-DMA EROM download routines will fill a total of 32 Kwords of EROM (addresses 0x8000 to 0xFFFF) unless terminated by INT3. DMA EROM download routines fill a total of 64 Kwords as YMAP4 first (addresses 0x0 to 0x7FFF) then YMAP3 (addresses 0x8000 to 0xFFFF), unless terminated by INT3. The DSP1620 boot routines execute at 1X the input clock (CKI) rate, unless the PLL routine is selected first.

The following is a brief description of the DSP1620 boot routines:

**1. Download to internal DPRAM from the PHIF16 port, 8-bit external mode, 8-bit logical transfers (0x001, 0x002, 0x003).**

These are single-byte transfers, either *Motorola* or *Intel* style. For *Motorola* style, the polarity of PDS can be selected. The routine builds 16-bit words from the incoming bytes. The least significant byte is always transferred first.

**2. Download to internal DPRAM from the PHIF16 port, 8-bit external mode, 16-bit logical transfers (0x0006, 0x0007, 0x0008).**

These are single-byte transfers, but the PHIF16 port builds the 16-bit word. The host toggles PBSEL when transferring the high (1) and low bytes (0) of the incoming word. Protocol can be selected between *Motorola* and *Intel* styles. For *Motorola* style, the polarity of PDS can also be selected.

**3. Download to internal DPRAM from the PHIF16 port, 16-bit external mode, non-DMA (0x0009, 0x000b, 0x0016, 0x0017).**

These are single-word transfers, either *Motorola* or *Intel* style. Most of these routines assume that the input word is not byte-swapped. For *Motorola* style, the polarity of PDS can be selected. Codes 0x16 and 0x17 byte swap the input word for *Motorola*-style interfaces.

**4. Download to internal DPRAM from the PHIF16 port, 16-bit external mode, DMA (0x000c, 0x000d, 0x000e, 0x001c, 0x001d).**

These routines are similar to #3, except that these are block transfers using the MIOU in a polled fashion. The block size is 64 words. The data loaded by the host should be in block sizes that are integer multiples of 64 words. If the last block of data is not 64 words long, it should be padded with zeros to ensure a 64-word block size. If an incomplete block has been transferred when the routine is terminated by the assertion of INT3, all of the data in this last block will be lost.

Codes 0x1c and 0x1d byte swap in input word for *Motorola*-style interfaces.

**5. Download to internal DPRAM from SIO, 8-bit mode (0x0010, 0x0011).**

Active or passive, SIO signaling is supported in LSB first mode. The routine builds 16-bit words from incoming data.

**6. Download to internal DPRAM from SIO, 16-bit mode (0x0012, 0x0013).**

Active or passive, SIO signaling is supported in LSB first mode.

**7. Download to internal DPRAM from I/O address 0x8000, 8- or 16-bit (0x0018, 0x0019).**

This routine performs consecutive reads of address 0x8000, presumably a memory-mapped peripheral. If 8-bit reads are performed, data must appear on DB[7:0], least significant byte first. The routine builds 16-bit words. Wait-states can be selected.

**8. Download to external ROM from PHIF16, 8-bit external mode, 8-bit logical transfers (0x0020, 0x0021, 0x0022).**

This routine is similar to #1. External wait-states can be selected.

**9. Download to external ROM from PHIF16, 8-bit mode, 16-bit logical transfers (0x0023, 0x0024, 0x0025).**

This routine is similar to #2. External wait-states can be selected.

**10. Download to external ROM from PHIF16, 16-bit external mode, non-DMA (0x0004, 0x0005, 0x0026, 0x0027, 0x0028).**

This routine is similar to #3. External wait-states can be selected. Codes 0x4 and 0x5 byte swap the input word for *Motorola* style interfaces.

**11. Download to external ROM from PHIF16, 16-bit external mode, DMA (0x000f, 0x002f).**

This routine is similar to #4. External wait-states can be selected. A total of 64 Kwords can be loaded, filling memory from 0x0 to 0x7FFF first, then 0x8000 to 0xFFFF. Codes 0xf and 0x2f byte swap the input word for *Motorola*-style interfaces.

## 7 DSP1620 Boot Routines (continued)

**12. Memory test routines (0x0030 to 0x003b, excluding 0x0036).**

There is a single boot routine to select every memory segment of the DSP1620 and test each segment for memory failures. The entire memory segment is tested, unless a failure is detected or the routine is terminated by INT3. RAM test routines will report the value 0x0fab when the test has completed successfully, and 0x0bad when a failure is detected. This result is loaded to **ar3** and can be written to the host processor through PHIF16 via the PHIF16 output routine (see #16, below), if immediately requested after the execution of the test routine.

For external memory segments, wait-states can be selected.

A CRC test is included to test the internal ROM content. Results are reported through JTAG (only).

**13. Memory checksum routines (0x002c, 0x002d, 0x002e).**

These routines perform a checksum on the entire DPRAM or EROM contents. The result is reported as a 32-bit result (written as high-half first, then low-half) via the PHIF16 in 16-bit *Intel* mode (only). The checksum of DPRAM can be performed on the entire 30 Kwords content, or optionally in 2 Kwords blocks. In either case, the host must read the results (two 16-bit words) of the checksum before the routine can continue (either to wait for the next routine selection, or to perform the checksum on the next 2 Kwords block). The checksum of EROM is always performed on a 32 Kwords segment, unless terminated by INT3. Premature termination will yield unpredictable results.

To execute the DPRAM checksum as 2 Kwords blocks, PB[7:6] must be written with a 01 during the selection of the DPRAM checksum routine (select byte = 0x6c). All other values in these bits will run this routine on the entire 30 Kwords DPRAM segment.

Wait-states can be enabled for the EROM checksum routines.

**14. XMAP select and execute routines (0x003c, 0x003d, 0x003e, 0x003f).**

These routines switch to XMAP1, 3, or 4 completely under software control. Execution then begins at address 0x0 in DPRAM for XMAP3 and 4. For XMAP1, execution begins at address 0x8000. For XMAP2, the boot routine clears LOWPR of the **alf** register then puts DSP1620 into sleep mode, waiting for the host to set EXM and issues a reset to the DSP.

For XMAP1 execution, a jump table is included as part of the ROM-coded boot routines. The user must place his/her interrupt vector table starting at address 0x8000 in this case. The jump table assumes that each interrupt vector (that is enabled) will be located at its vector address offset by 0x8000.

**15. PHIF16 output routine (0x0015).**

The results of all the memory test routines, with the exception of the CRC and checksum routines, can be output via the PHIF16 in 16-bit mode. All 16-bit modes are supported, each mode being selected by the value on PB[7:6] (see bit field listing, above). This routine writes a single 16-bit value, and waits for the host to read it. The host must read the PHIF16 port or issue an INT3 for the boot routines to continue (program flow is stalled until the DSP writes out the value).

**16. PLL enable routine (0x001e, 0x001f).**

This routine enables the PLL with one of three predetermined values of M and N, or it disables the PLL. Selection of multiplier is made by the value of PB[7:6] and can be 10X, 5X, or 2X the input clock, CKI. In addition, the PLL can be disabled through this routine (PB[7:6] = 00). Please note that there are different routines for 3 V and 5 V operation.

The PLL is not automatically disabled when the boot routines are terminated. Once enabled, the user can disable the PLL through the PLL boot routine, DSP1620 program instructions, or RESET if so desired.

The source code for the DSP1620 boot routines is available from Lucent Technologies Technical Support Engineering at:

http://www.lucent.com/micro/wam/tse

**Note**: As previously stated, the boot routine selection process is always performed as an 8-bit *Intel* style input transfer. This is done so that a single strobe executes the transfer. For the user interfacing the DSP1620 to a *Motorola* or *Motorola* compatible processor, some external logic is required for proper operation.

It is recommended that the PDS output of the *Motorola* device be gated with a control signal (an IOBIT or address bit). For example, by ANDing PDS with this control bit, PDS can be blocked from the DSP1620's PODS input (in *Intel* mode) during the byte select process when the control bit is clear. Once a *Motorola*-style boot routine has been selected (either PDS active-high or active-low), the control bit must be set to allow PDS to effect the transfer.

# 8 Device Characteristics

## 8.1 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.

- Voltage Range on $V_{DD}$ with Respect to Ground Using Devices Designed for 5 V Operation .......... –0.5 V to +7 V
- Voltage Range on $V_{DD}$ with Respect to Ground Using Devices Designed for 3 V Operation ....... –0.5 V to +4.6 V
- Voltage Range on Any Signal Pin ........ $V_{SS}$ – 0.5 V to $V_{DD}$ + 0.5 V
- Power Dissipation ........ 1 W
- Junction Temperature ($T_J$) ........ –40 °C to +125 °C
- Storage Temperature Range ........ –65 °C to +150 °C

## 8.2 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 DSP1620 is greater than 2000 V.

## 8.3 Recommended Operating Conditions

**Table 74. Recommended Operating Conditions**

| Maximum Instruction Rate (MIPS) | Device Speed (ns) | Input Clock | Supply Voltage $V_{DD}$ (V) | | Ambient Temperature $T_A$ (°C) | |
|---|---|---|---|---|---|---|
| | | | Min | Max | Min | Max |
| 90 | 11.1 | CMOS | 3.0 | 3.6 | –40 | 85 |
| 100 | 10.0 | CMOS | 3.0 | 3.6 | –40 | 85 |
| 120 | 8.3 | CMOS | 3.0 | 3.6 | –40 | 85 |
| 90 | 11.1 | CMOS | 4.75 | 5.25 | –40 | 85 |

The ratio of the instruction cycle rate to the input clock frequency is 1:1 without the PLL (referred to as 1X operation) and M/(2N) with the PLL selected (see Section 4.16). **Device speeds greater than 50 MIPS do not support 1X operation; use the PLL.**

# 8 Device Characteristics (continued)

## 8.4 Package Thermal Considerations

The recommended operating temperature specified above 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.

- 132-pin BQFP Maximum Junction Temperature ($T_J$) .......... 110 °C
- 132-pin BQFP Maximum Thermal Resistance in Still-Air-Ambient ($\Theta_{JA}$) .......... 42 °C/W
- 132-pin BQFP Maximum Thermal Resistance, Junction to Top Dead Center ($\Theta_{J\text{-}TDC}$) .......... 9 °C/W
- 144-pin TQFP Maximum Junction Temperature ($T_J$) .......... 110 °C
- 144-pin TQFP Maximum Thermal Resistance in Still-Air-Ambient ($\Theta_{JA}$) .......... 30 °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 110 °C. Otherwise, the device performance is affected adversely.**

# 9 Electrical Characteristics and Requirements

The following electrical characteristics are preliminary and are subject to change. 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 Section 8.3, Recommended Operating Conditions.

**Table 75. Electrical Characteristics and Requirements**

| Parameter | Symbol | Min | Max | Unit |
|---|---|---|---|---|
| Input Voltage: | | | | |
| Low | $V_{IL}$ | –0.3 | 0.3 * $V_{DD}$ | V |
| High | $V_{IH}$ | 0.7 * $V_{DD}$ | $V_{DD}$ + 0.3 | V |
| Input Current (except TMS, TDI): | | | | |
| Low ($V_{IL}$ = 0 V, $V_{DD}$ = 5.25 V) | $I_{IL}$ | –5 | — | µA |
| High ($V_{IH}$ = 5.25 V, $V_{DD}$ = 5.25 V) | $I_{IH}$ | — | 5 | µA |
| Input Current (TMS, TDI): | | | | |
| Low ($V_{IL}$ = 0 V, $V_{DD}$ = 5.25 V) | $I_{IL}$ | –100 | — | µA |
| High ($V_{IH}$ = 5.25 V, $V_{DD}$ = 5.25 V) | $I_{IH}$ | — | 5 | µA |
| Output Low Voltage: | | | | |
| Low ($I_{OL}$ = 2.0 mA) | $V_{OL}$ | — | 0.4 | V |
| Low ($I_{OL}$ = 50 µA) | $V_{OL}$ | — | 0.2 | V |
| Output High Voltage: | | | | |
| High ($I_{OH}$ = –2.0 mA) | $V_{OH}$ | $V_{DD}$ – 0.7 | — | V |
| High ($I_{OH}$ = –50 µA) | $V_{OH}$ | $V_{DD}$ – 0.2 | — | V |
| Output 3-State Current: | | | | |
| Low ($V_{DD}$ = 5.25 V, $V_{IL}$ = 0 V) | $I_{OZL}$ | –10 | — | µA |
| High ($V_{DD}$ = 5.25 V, $V_{IH}$ = 5.25 V) | $I_{OZH}$ | — | 10 | µA |
| Input Capacitance | $C_I$ | — | 5 | pF |

**Table 76. Electrical Requirements for Mask-Programmable Input Clock Options**

| Parameter | Symbol | Min | Max | Unit |
|---|---|---|---|---|
| CKI CMOS Level Input Voltage: | | | | |
| Low | $V_{IL}$ | –0.3 | 0.3 * $V_{DD}$ | V |
| High | $V_{IH}$ | 0.7 * $V_{DD}$ | $V_{DD}$ + 0.3 | V |

## 9 Electrical Characteristics and Requirements (continued)

**Table 77. PLL Electrical Specifications, VCO Frequency Ranges**

| Parameter | Symbol | Min | Max | Unit |
|---|---|---|---|---|
| VCO* Frequency Range ($V_{DD}$ = 3.3 V ± 0.3 V) | $f_{VCO}$ | 90 | 240 | MHz |
| VCO* Frequency Range ($V_{DD}$ = 5 V ± 5%) | $f_{VCO}$ | 90 | 180 | MHz |
| Input Jitter at CKI | — | — | 200 | ps-rms |

* The M and N counter values in the **pllc** register must be set so that the VCO operates in the appropriate range. Choose the lowest value of N and then the appropriate value of M for $f_{internal\ clock} = f_{CKI} \times (M/(2N)) = f_{VCO}/2$.

**Table 78. PLL Electrical Specifications and pllc Register Settings**

| M | $V_{DD}$ | pllc13 (ICP) | pllc12 (SEL5V) | pllc[11:8] (LF[3:0]) | Typical Lock-In Time (µs)* |
|---|---|---|---|---|---|
| 23—24 | 3.3 V ± 0.3 V | 1 | 0 | 1011 | 30 |
| 21—22 | 3.3 V ± 0.3 V | 1 | 0 | 1010 | 30 |
| 19—20 | 3.3 V ± 0.3 V | 1 | 0 | 1001 | 30 |
| 16—18 | 3.3 V ± 0.3 V | 1 | 0 | 1000 | 30 |
| 12—15 | 3.3 V ± 0.3 V | 1 | 0 | 0111 | 30 |
| 8—11 | 3.3 V ± 0.3 V | 1 | 0 | 0110 | 30 |
| 2—7 | 3.3 V ± 0.3 V | 1 | 0 | 0100 | 30 |
| 19—20 | 5 V ± 5% | 1 | 1 | 1110 | 30 |
| 17—18 | 5 V ± 5% | 1 | 1 | 1101 | 30 |
| 16 | 5 V ± 5% | 1 | 1 | 1100 | 30 |
| 14—15 | 5 V ± 5% | 1 | 1 | 1011 | 30 |
| 12—13 | 5 V ± 5% | 1 | 1 | 1010 | 30 |
| 10—11 | 5 V ± 5% | 1 | 1 | 1001 | 30 |
| 8—9 | 5 V ± 5% | 1 | 1 | 1000 | 30 |
| 7 | 5 V ± 5% | 1 | 1 | 0111 | 30 |
| 5—6 | 5 V ± 5% | 1 | 1 | 0110 | 30 |
| 2—4 | 5 V ± 5% | 1 | 1 | 0101 | 30 |

* Lock-in time represents 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 1X CKI input clock or from the slow ring oscillator while the PLL is locking. Completion of the lock-in interval is indicated by assertion of the LOCK flag.

## 9 Electrical Characteristics and Requirements (continued)

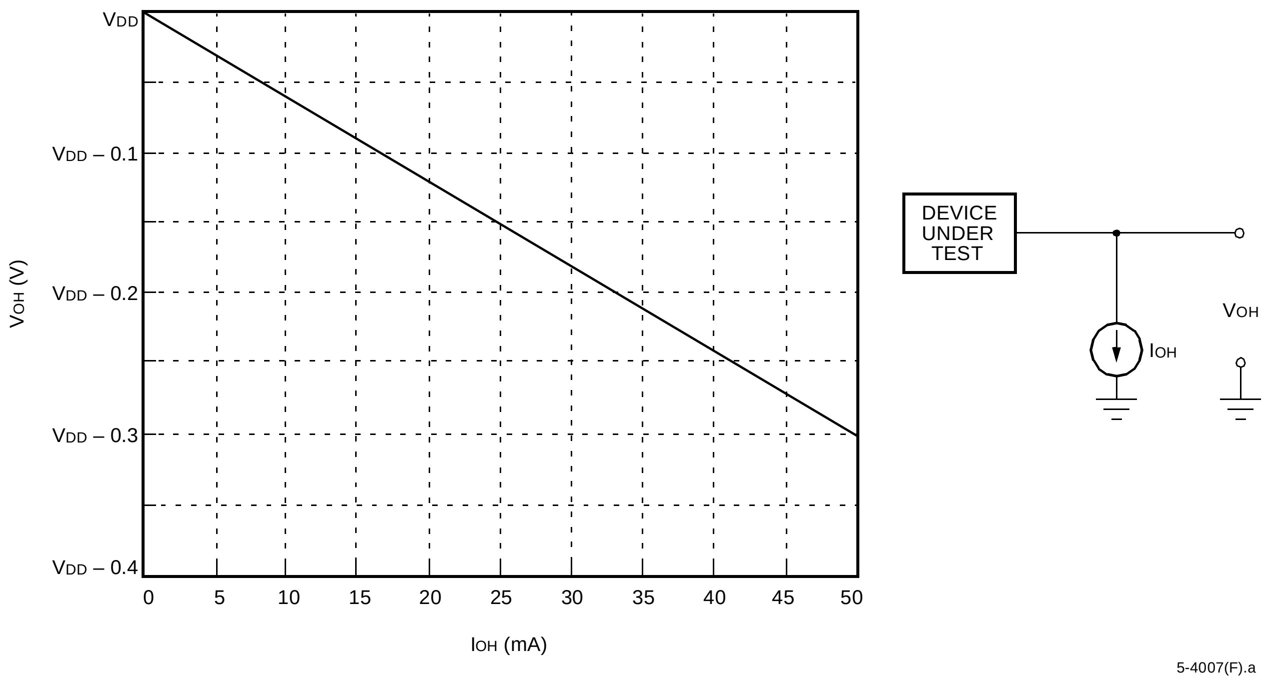


**Figure 16. Plot of $V_{OH}$ vs. $I_{OH}$ Under Typical Operating Conditions**

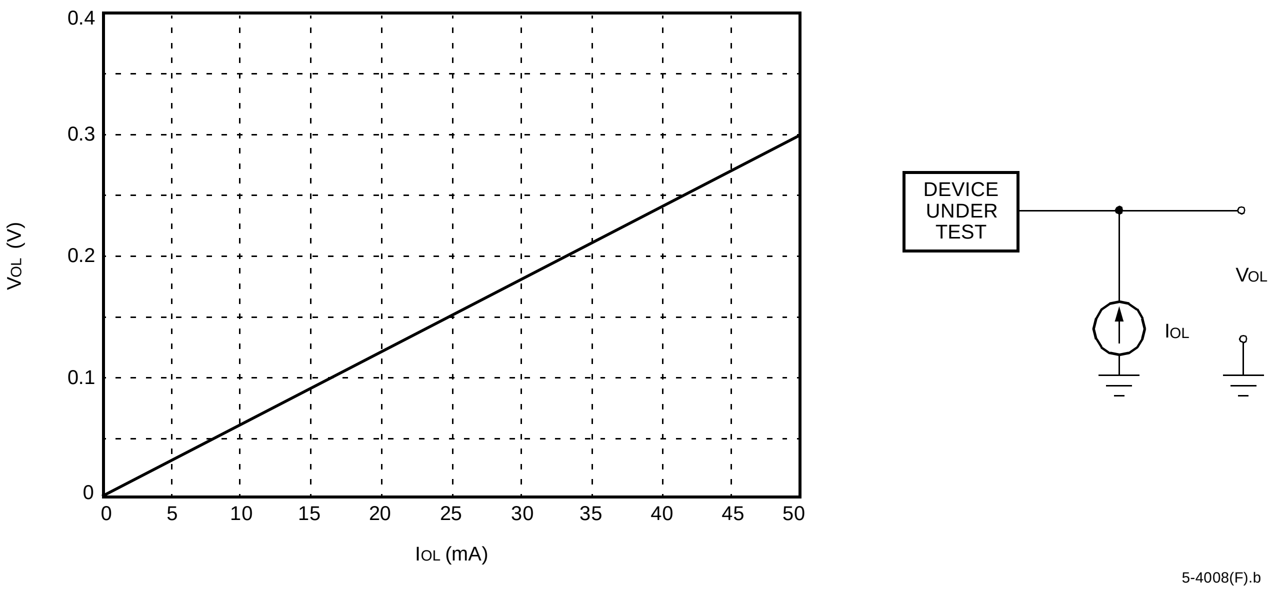


**Figure 17. Plot of $V_{OL}$ vs. $I_{OL}$ Under Typical Operating Conditions**

## 9 Electrical Characteristics and Requirements (continued)

### 9.1 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. The following electrical characteristics are preliminary and are subject to change.

**Table 79. Power Dissipation and Wake-Up Latency**

| **Operating Mode** (Unused inputs at $V_{DD}$ or $V_{SS}$) | **Typical Power Dissipation (mW)** | | | | **Wake-Up Latency** | | | |
|---|---|---|---|---|---|---|---|---|
| | **I/O Units ON powerc[7:4] = 0x0** | | **I/O Units OFF powerc[7:4] = 0xf** | | **(PLL Not Used During Wake State)** | | **(PLL Used During Wake State)** | |
| | **5 V** | **3 V** | **5 V** | **3 V** | **5 V** | **3 V** | **5 V** | **3 V** |
| Normal Operation **ioc** = 0x0180 **PLL Disabled** CKI & CKO = 40 MHz | | | | | | | | |
| CMOS | 296 | 116 | 282 | 110 | — | | — | |
| CKI & CKO = 0 MHz | | | | | | | | |
| CMOS | 0.55 | 0.2 | 0.55 | 0.2 | — | | — | |
| Normal Operation **ioc** = 0x0180 **PLL Enabled** **pllc** = 0xFC0E CKI = 10 MHz CKO = 40 MHz | | | | | | | | |
| CMOS | 304 | 118 | 289 | 113 | — | | — | |
| **Power Management Modes** CKO = 40 MHz | | | | | | | | |
| Standard Sleep, External Interrupt **alf**[15] = 1, **ioc** = 0x0180 **PLL Disabled During Sleep** | | | | | | | | |
| CMOS | 44 | 16 | 36 | 12 | 3T* | | 3T* + $t_L$† | |
| Standard Sleep, External Interrupt **alf**[15] = 1, **ioc** = 0x0180 **PLL Enabled During Sleep** | | | | | | | | |
| CMOS | 52 | 20 | 40 | 16 | — | | 3T* | |
| Sleep with Slow Internal Clock **powerc**[15:14] = 01, **alf**[15] = 1, **ioc** = 0x0180 **PLL Disabled During Sleep** | | | | | | | | |
| CMOS | 1.25 | 0.51 | 1.25 | 0.51 | 1.5 µs | 5.0 µs | 1.5 µs + $t_L$† | 5.0 µs + $t_L$† |
| Sleep with Slow Internal Clock **powerc**[15:14] = 01, **alf**[15] = 1, **ioc** = 0x0180 **PLL Enabled During Sleep** | | | | | | | | |
| CMOS | 8.5 | 3.3 | 8.5 | 3.3 | — | | 1.5 µs | 5.0 µs |

* T = CKI clock cycle for 1X input clock option or T = CKI clock cycle divided by M/(2N) for PLL clock option (see Section 4.16).
† $t_L$ = PLL lock time (see Table 78).

## 9 Electrical Characteristics and Requirements (continued)

**Table 79. Power Dissipation and Wake-Up Latency** (continued)

| **Operating Mode** (Unused inputs at $V_{DD}$ or $V_{SS}$) | **Typical Power Dissipation (mW)** | | | | **Wake-Up Latency** | | | |
|---|---|---|---|---|---|---|---|---|
| | **I/O Units ON powerc[7:4] = 0x0** | | **I/O Units OFF powerc[7:4] = 0xf** | | **(PLL Not Used During Wake State)** | | **(PLL Used During Wake State)** | |
| | **5 V** | **3 V** | **5 V** | **3 V** | **5 V** | **3 V** | **5 V** | **3 V** |
| Software Stop **powerc**[15:12] = 0011 **PLL Disabled During STOP** | | | | | | | | |
| CMOS | 0.55 | 0.2 | 0.55 | 0.2 | 3T* | | 3T* + $t_L$† | |
| Hardware Stop (STOP = $V_{SS}$) **powerc**[15:12] = 0000 **PLL Disabled During STOP** | | | | | | | | |
| CMOS | 0.55 | 0.2 | 0.55 | 0.2 | 3T* | | — | |
| Hardware Stop (STOP = $V_{SS}$) **powerc**[15:12] = 0000 **PLL Enabled During STOP** | | | | | | | | |
| CMOS | 7 | 3 | 7 | 3 | 3T* | | 3T* | |

* T = CKI clock cycle for 1X input clock option or T = CKI clock cycle divided by M/(2N) for PLL clock option (see Section 4.16).
† $t_L$ = PLL lock time (see Table 78).

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. Although input and I/O buffers may be left untied (since the input voltage levels of the input and I/O buffers are designed to remain at full CMOS levels when not driven), it is still recommended that unused input and I/O pins be tied to $V_{SS}$ or $V_{DD}$ through a 10 kΩ resistor to avoid application ambiguities. Further, if I/O pins are tied high or low, they should be pulled fully to $V_{SS}$ or $V_{DD}$.

**WARNING: The device needs to be clocked for at least six CKI cycles during reset after powerup. Otherwise, high currents may flow.**

## 10 Timing Characteristics and Requirements for 5 V Operation

The following timing characteristics and requirements are preliminary information and are subject to change. 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 Section 8.3.)

$V_{DD}$ = 5 V ± 5%, $V_{SS}$ = 0 V (See Section 8.3.)

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: | 0.03 ns/pF ≤ $dt/dC_L$ ≤ 0.06 ns/pF for 10 ≤ $C_L$ ≤ 100 pF |
| CKO: | 0.06 ns/pF ≤ $dt/dC_L$ ≤ 0.12 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 is 30 pF instead of 50 pF, the derating for a rising edge is (30 – 50) pF x 0.06 ns/pF = 1.2 ns **less** than the specified rise time or delay that includes a rise time.

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):

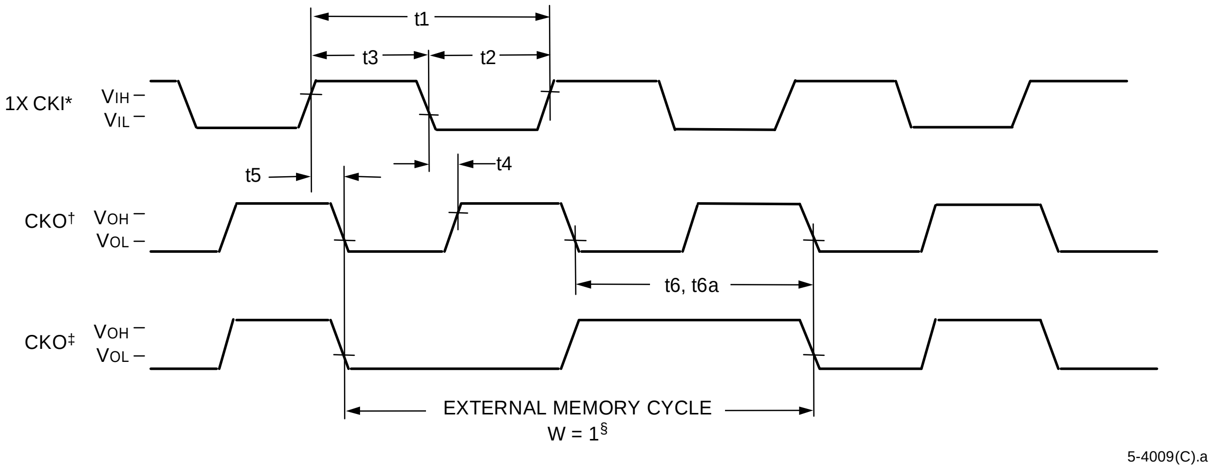
- $C_{LOAD}$ = 50 pF
- Timing reference levels for delays = $V_{IH}$, $V_{IL}$
- 3-state delays measured to the high-impedance state of the output driver

For the timing diagrams, see Table 76 for input clock requirements.

Unless otherwise noted, CKO in the timing diagrams is the free-running CKO.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

### 10.1 DSP Clock Generation



* See Table 76 for input clock electrical requirements.
† Free-running clock.
‡ Wait-stated clock (see Table 41).
§ W = number of wait-states.

**Figure 18. I/O Clock Timing Diagram**

**Table 80. 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 for 1X input clock option, and memory hold time is tested at 0.1 s.

**Table 81. Timing Characteristics for Input Clock and Output Clock**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t4 | Clock Out High Delay | — | 8 | ns |
| t5 | Clock Out Low Delay (high to low) | — | 8 | ns |
| t6 | Clock Out Period (low to low) | T* | — | ns |
| t6a | Clock Out Period with SLOWCKI Bit Set in **powerc** Register (low to low) | 0.74 | 1.6 | µs |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

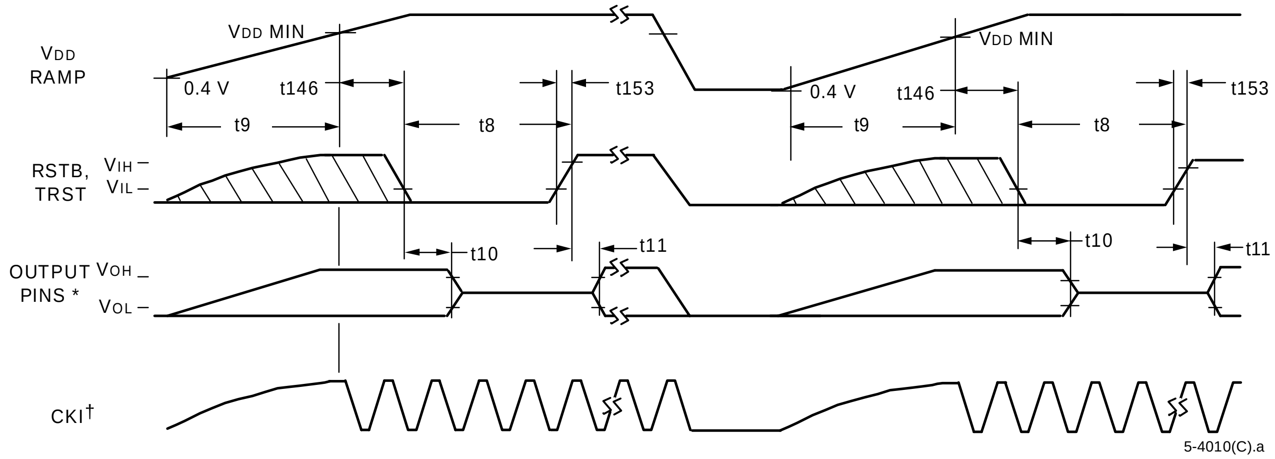
### 10.2 Reset Circuit

The DSP1620 has two external reset pins: RSTB and TRST. At initial powerup, or if the supply voltage falls below VDD MIN* and a device reset is required, both TRST and RSTB must be asserted to initialize the device. Figure 19 shows two separate events:

1. Chip reset at initial powerup.
2. Chip reset following a drop in power supply.

**Note: The TRST pin must be asserted even if the application does not use the JTAG controller.**

* See Table 74, Recommended Operating Conditions.



* When the level of RSTB = 0 and INT0 = 1, all output and bidirectional pins (except TDO, which 3-states by JTAG control) are put in a 3-state condition. When the level of RSTB = 0 and INT0 = 0, EROM, ERAMHI, ERAMLO, IO, and RWN outputs remain high, and CKO remains a free-running clock.

† SeeTable 76 for input clock electrical requirements.

**Figure 19. Powerup Reset and Chip Reset Timing Diagram**

**Table 82. Timing Requirements for Powerup Reset and Chip 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) | — | 95 | ns |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

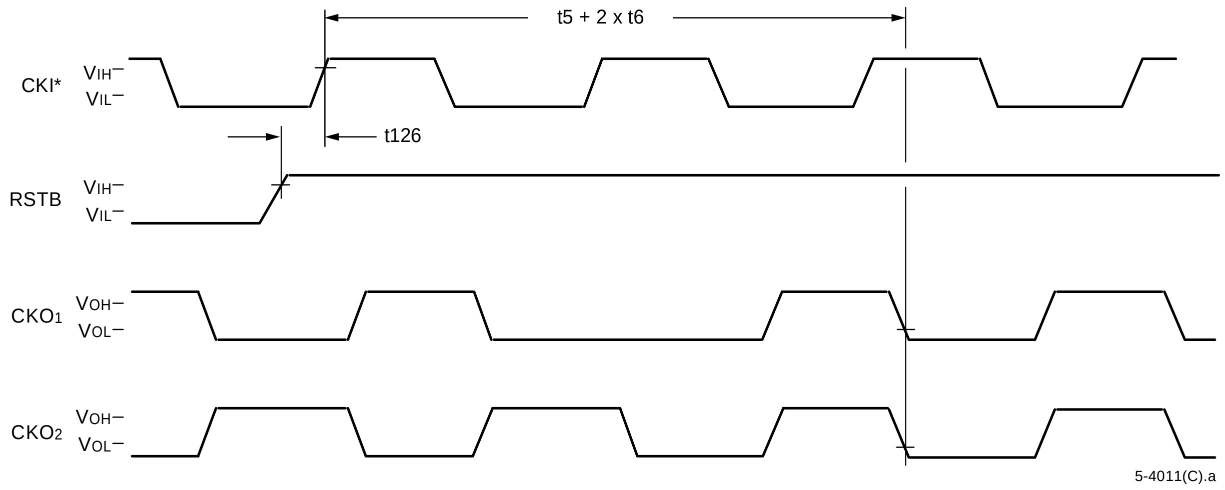
**Table 83. Timing Characteristics for Powerup Reset and Chip Reset**

| 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 may flow.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

### 10.3 Reset Synchronization



* See Table 76 for input clock electrical requirements.

Notes:
CKO1 and CKO2 are two possible CKO states before reset. CKO is free-running.

If the rising edge of RSTB (low to high) is captured instead by the falling edge of CKO (high to low), CKO and CKI will be in-phase at t5 + 2 x t6.

**Figure 20. Reset Synchronization Timing**

**Table 84. Timing Requirements for Reset Synchronization Timing**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t126 | Reset Setup (high to high) | 1.5 | T/2 – 1* | ns |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

### 10.4 JTAG I/O Specifications

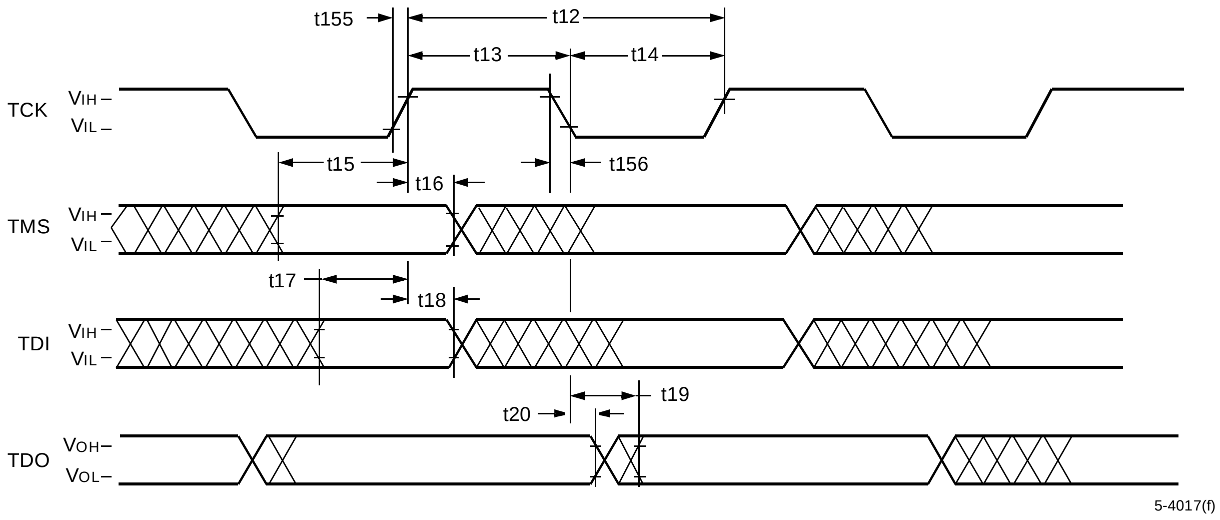


**Figure 21. JTAG Timing Diagram**

**Table 85. Timing Requirements for JTAG Input/Output**

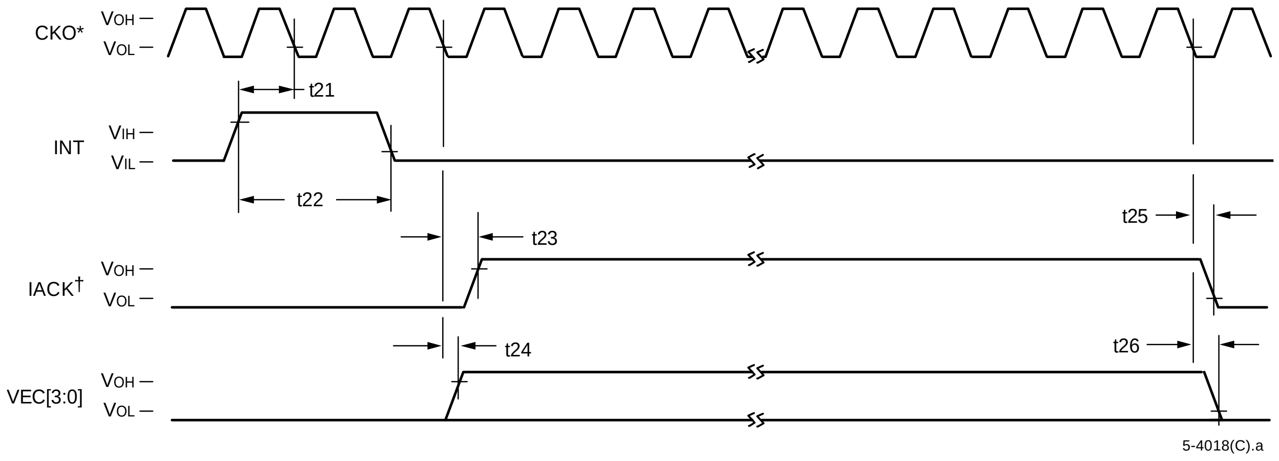
| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t12 | TCK Period (high to high) | 50 | — | ns |
| t13 | TCK High Time (high to low) | 22.5 | — | ns |
| t14 | TCK Low Time (low to high) | 22.5 | — | ns |
| t155 | TCK Rise Transition Time (low to high) | 0.6 | — | V/ns |
| 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) | 2 | — | ns |
| t17 | TDI Setup Time (valid to high) | 7.5 | — | ns |
| t18 | TDI Hold Time (high to invalid) | 2 | — | ns |

**Table 86. Timing Characteristics for JTAG Input/Output**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t19 | TDO Delay (low to valid) | — | 19 | ns |
| t20 | TDO Hold (low to invalid) | 0 | — | ns |

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

### 10.5 Interrupt



* CKO is free-running.
† IACK assertion is guaranteed to be enclosed by VEC[3:0] assertion.

**Figure 22. Interrupt Timing Diagram**

**Table 87. Timing Requirements for Interrupt**

**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) | 15 | — | ns |
| t22 | INT Assertion Time (high to low) | 2T* | — | ns |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

**Table 88. Timing Characteristics for Interrupt**

**Note:** Interrupt is asserted during an interruptible instruction and no other pending interrupts.

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t23 | IACK Assertion Time (low to high) | — | T/2 + 7.5* | ns |
| t24 | VEC Assertion Time (low to high) | — | 9.5 | ns |
| t25 | IACK Invalid Time (low to low) | — | 7.5 | ns |
| t26 | VEC Invalid Time (low to low) | — | 9.5 | ns |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

### 10.6 Bit Input/Output (BIO)

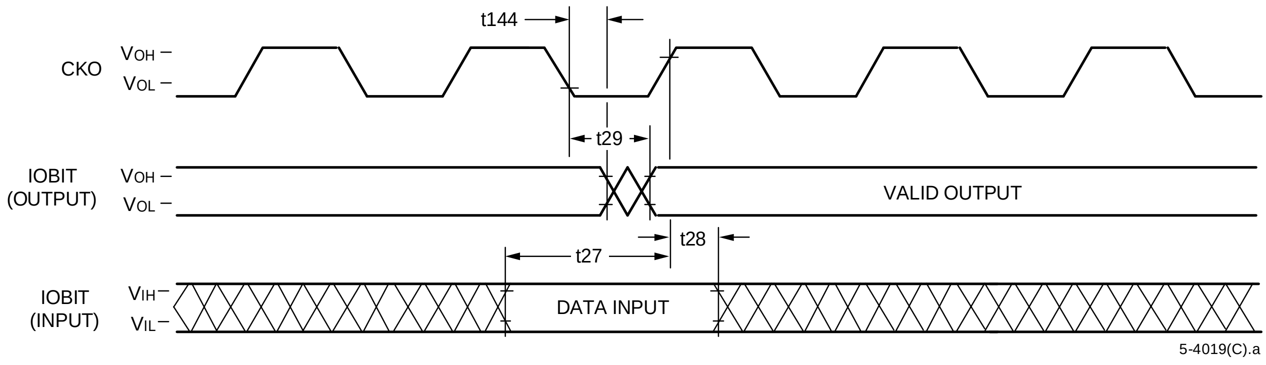


**Figure 23. Write Outputs Followed by Read Inputs (cbit = Immediate; a1 = sbit)**

**Table 89. Timing Requirements for BIO Input Read**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t27 | IOBIT Input Setup Time (valid to high) | 12 | — | ns |
| t28 | IOBIT Input Hold Time (high to invalid) | 0 | — | ns |

**Table 90. Timing Characteristics for BIO Output**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t29 | IOBIT Output Valid Time (low to valid) | — | 7.5 | ns |
| t144 | IOBIT Output Hold Time (low to invalid) | 1 | — | ns |

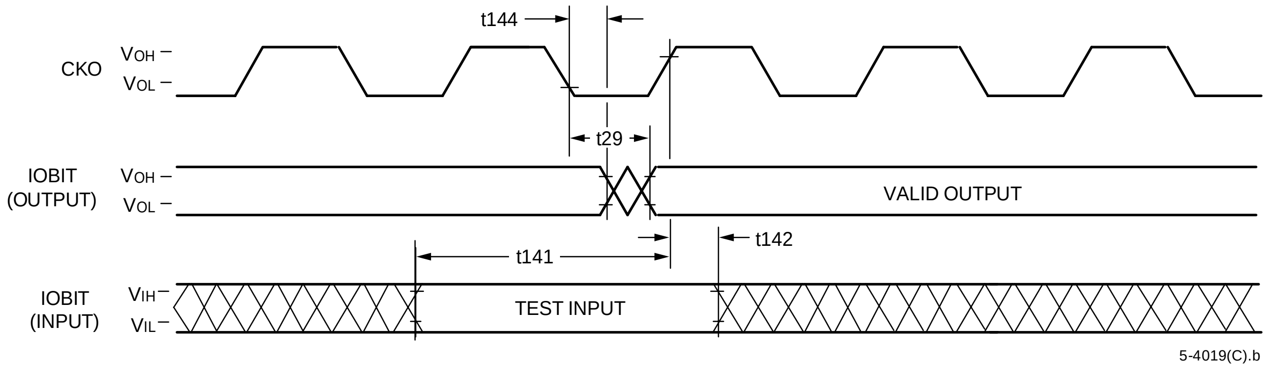


**Figure 24. Write Outputs and Test Inputs (cbit = Immediate)**

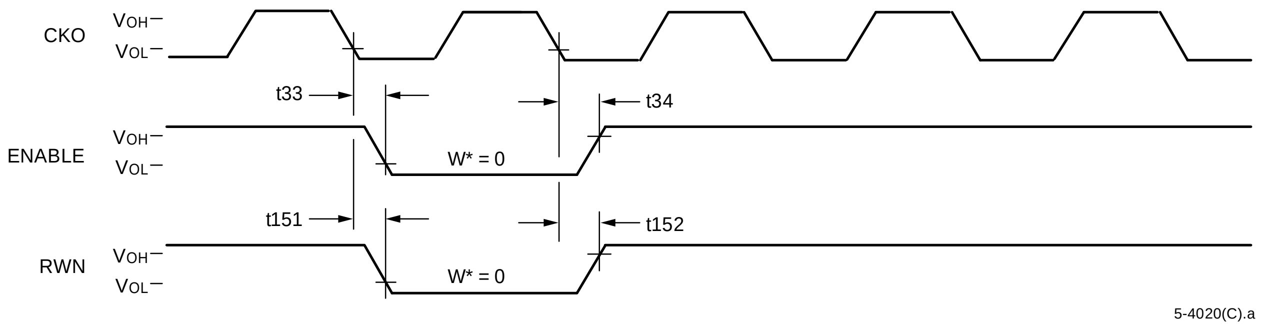
**Table 91. Timing Requirements for BIO Input Test**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t141 | IOBIT Input Setup Time (valid to high) | 13 | — | ns |
| t142 | IOBIT Input Hold Time (high to invalid) | 0 | — | ns |

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

### 10.7 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 *DSP1620 Digital Signal Processor* Information Manual for a detailed description of the external memory interface including other functional diagrams. The term ENABLE refers to EROM, ERAMX, IO, ERAMHI, and ERAMLO.



* W = number of wait-states.

**Figure 25. Enable Transition Timing**

**Table 92. Enable Transition Timing When Memory Enables and RWN Are Not Delayed (When ioc DENB[4:0] = 0x00 and ioc RWNADV = 0x1)**

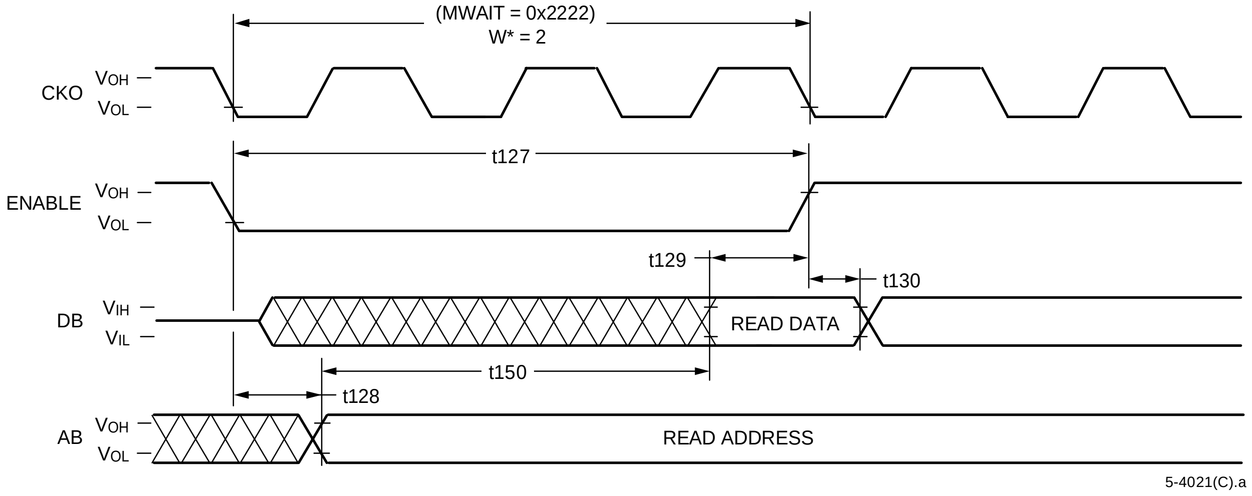
| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t33 | CKO to ENABLE Active (low to low) | 0 | 7 | ns |
| t34 | CKO to ENABLE Inactive (low to high) | –1 | 6 | ns |
| t151 | CKO to RWN (RWNADV = 0x1) Active (low to low) | 0 | 7 | ns |
| t152 | CKO to RWN (RWNADV = 0x1) Inactive (low to high) | –1 | 6 | ns |

**Table 93. Timing Characteristics for Delayed External Memory Enables (When ioc DENB[4:0] = 0x1F)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t33 | CKO to Delayed ENABLE Active (low to low) | T/2* | T/2 + 7* | ns |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)



* W = number of wait-states.

**Figure 26. External Memory Data Read Timing Diagram**

**Table 94. Timing Characteristics for External Memory Access**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t127 | Enable Width (low to high) | T(1 + W) – 4* | — | ns |
| t128 | Address Valid (enable low to valid) | — | 2 | ns |

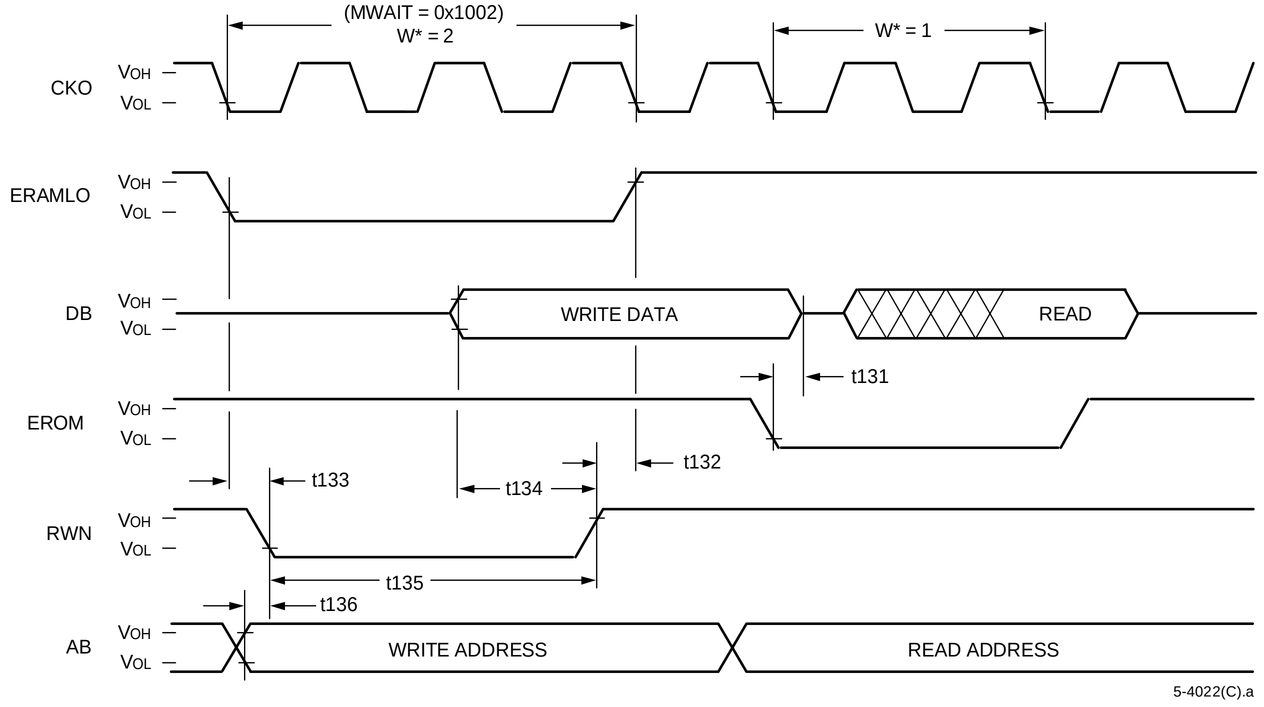
* T = internal clock period, set by CKI or by CKI and the PLL parameters.

**Table 95. Timing Requirements for External Memory Read (EROM, ERAMHI, I/O, ERAMLO)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t129 | Read Data Setup (valid to enable high) | 11 | — | ns |
| t130 | Read Data Hold (enable high to hold) | 0 | — | ns |
| t150 | External Memory Access Time (valid to valid) | — | T(1 + W) – 12* | ns |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)



* W = number of wait-states.

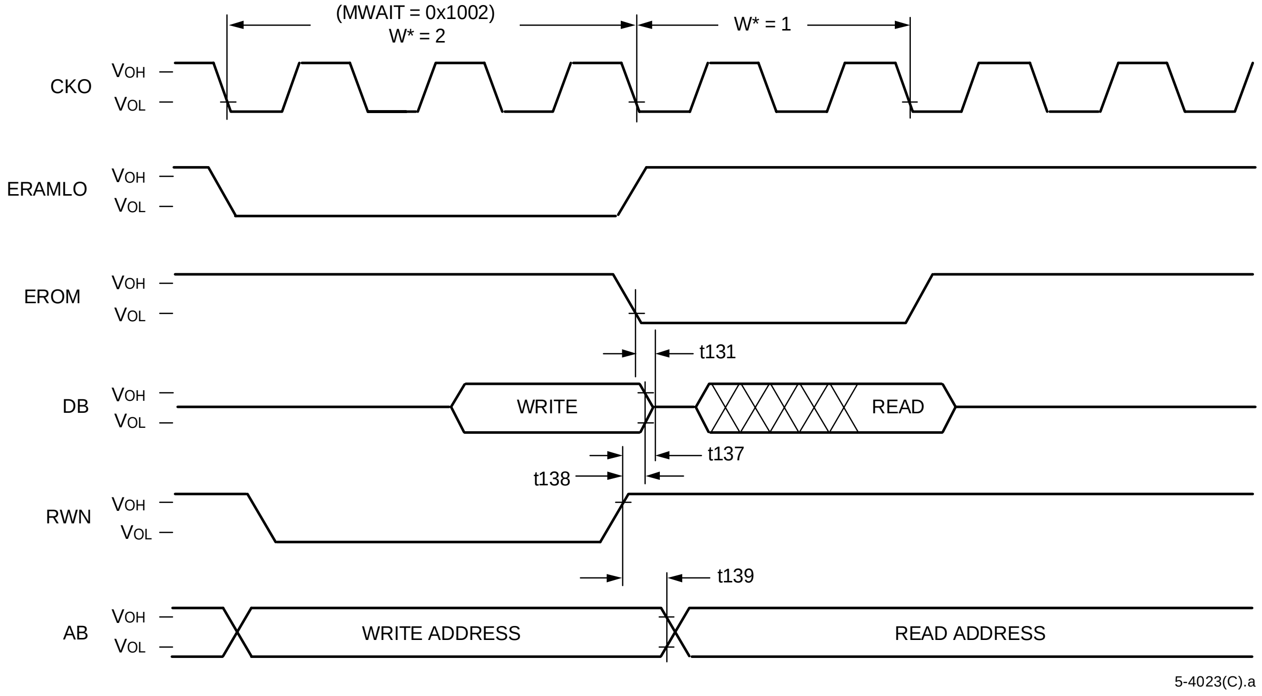
**Figure 27. External Memory Data Write Timing Diagram**

**Table 96. Timing Characteristics for External Memory Data Write (All Enables)***

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t131 | Write Overlap (enable low to 3-state) | — | 0 | ns |
| t132 | RWN Advance (RWN high to enable high) | 0 | 3 | ns |
| t133 | RWN Delay (enable low to RWN low) | 0 | 3 | ns |
| t134 | Write Data Setup (data valid to RWN high) | T(1 + W)/2 – 3† | — | ns |
| t135 | RWN Width (low to high) | T(1 + W) – 4† | — | ns |
| t136 | Write Address Setup (address valid to RWN low) | 0 | — | ns |

* **ioc** RWNADV = 0x0.
† T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)



* W = number of wait-states.

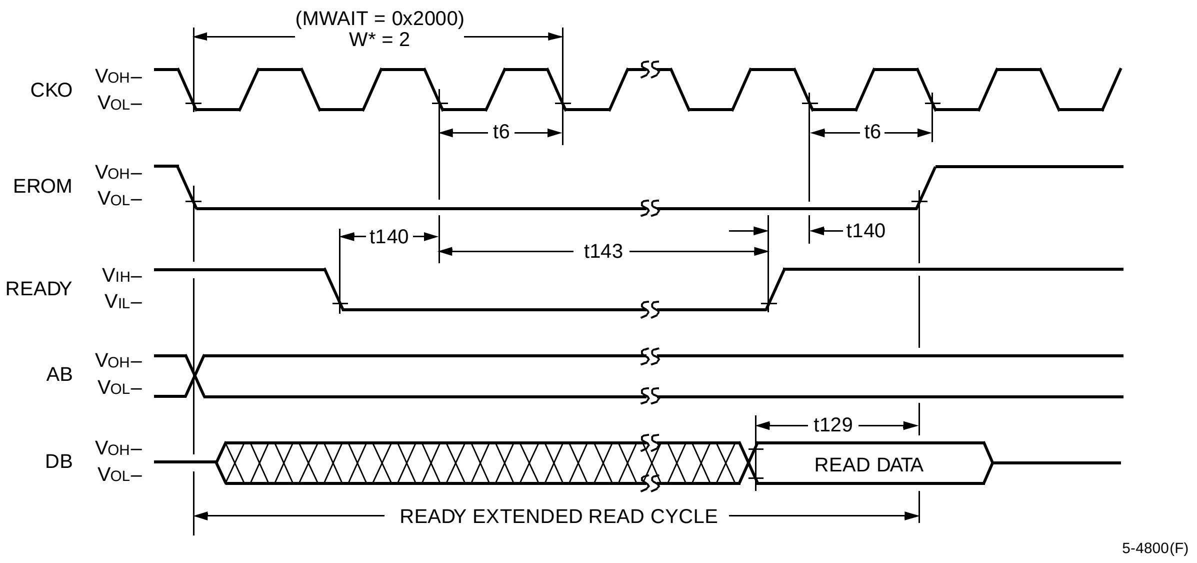
**Figure 28. Write Cycle Followed by Read Cycle**

**Table 97. Timing Characteristics for Write Cycle Followed by Read Cycle***

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t131 | Write Overlap (enable low to 3-state) | — | 0 | ns |
| t137 | Write Data 3-state (RWN high to 3-state) | — | 2 | ns |
| t138 | Write Data Hold (RWN high to data hold) | 0 | — | ns |
| t139 | Write Address Hold (RWN high to address hold) | 0 | — | ns |

* **ioc** RWNADV = 0x0.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)



* W = number of wait-states.

**Figure 29. READY Extended Read Cycle Timing**

**Table 98. Timing Requirements for READY Extended Read Cycle Timing**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t6 | Clock Out Period (low to low) | T* | — | ns |
| t129 | Read Data Setup (valid to enable high) | 11 | — | ns |
| t140 | Ready Setup (valid to CKO low) | 0 | — | ns |
| t143 | Ready Hold (CKO low to invalid) | 8 | — | ns |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

### 10.8 PHIF16 Specifications

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.

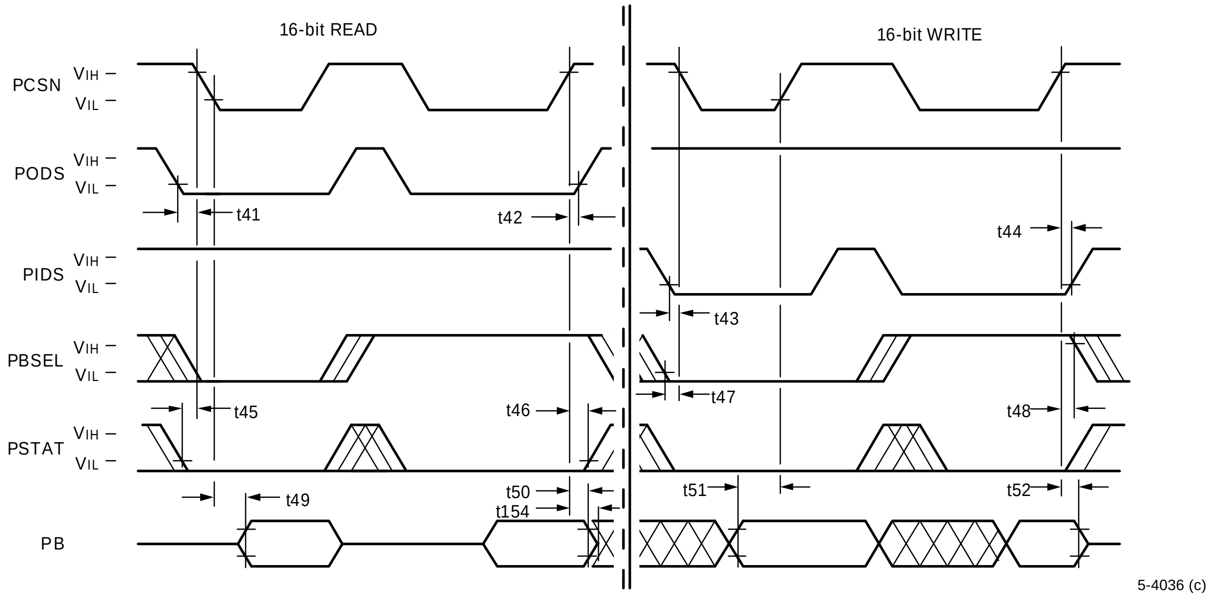


**Figure 30. PHIF16 *Intel* Mode Signaling (Read and Write) Timing Diagram**

**Table 99. 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) | 4.5 | — | ns |
| t46* | PCSN to PSTAT Hold (high to invalid) | 0 | — | ns |
| t47* | PBSEL to PCSN Setup (valid to low) | 4.5 | — | ns |
| t48* | PCSN to PBSEL Hold (high to invalid) | 0 | — | ns |
| t51* | PB Write to PCSN Setup (valid to high) | 7.5 | — | ns |
| t52* | PCSN to PB Write Hold (high to invalid) | 4 | — | ns |

* This timing diagram for the PHIF16 port 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. For example, the timing requirement referenced to PCSN going low, t45, should be referenced to PODS going low, if PODS goes low after PCSN. 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. All requirements referenced to PCSN apply to PIDS or PODS if either is the controlling signal.

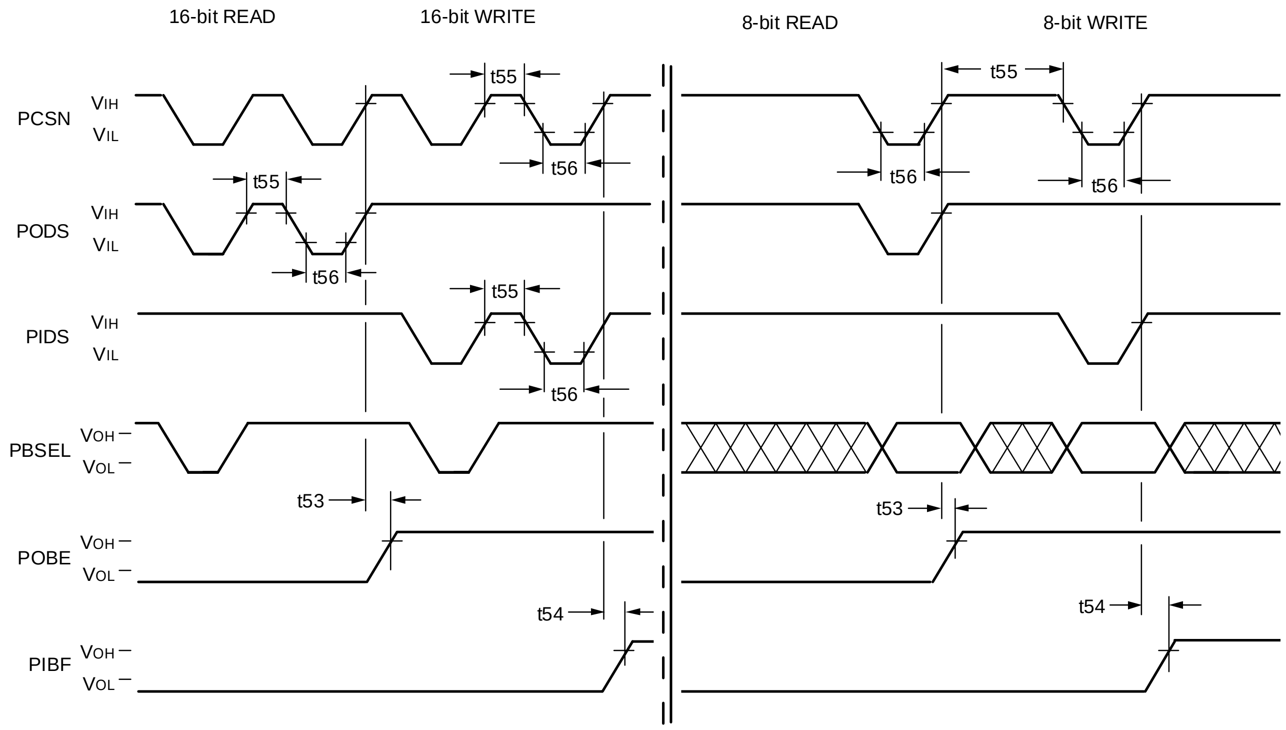
**Table 100. Timing Characteristics for PHIF16 *Intel* Mode Signaling (Read and Write)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t49† | PCSN to PB Read (low to valid) | — | 13 | ns |
| t50† | PCSN to PB Read Hold (high to invalid) | 0 | — | ns |
| t154 | PCSN to PB Read 3-state (high to 3-state) | — | 9 | ns |

† This timing diagram for the PHIF16 port 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. For example, the timing requirement referenced to PCSN going low, t49, should be referenced to PODS going low, if PODS goes low after PCSN. 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. All requirements referenced to PCSN apply to PIDS or PODS, if PIDS or PODS is the controlling signal.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

Assuming an 8-bit external interface:



5-4037(C).a

**Figure 31. PHIF16 *Intel* Mode Signaling (Pulse Period and Flags) Timing Diagram**

**Table 101. 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) | 15 | — | ns |
| t56 | PCSN/PODS/PIDS Pulse Width (low to high) | 15 | — | ns |

**Table 102. 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.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

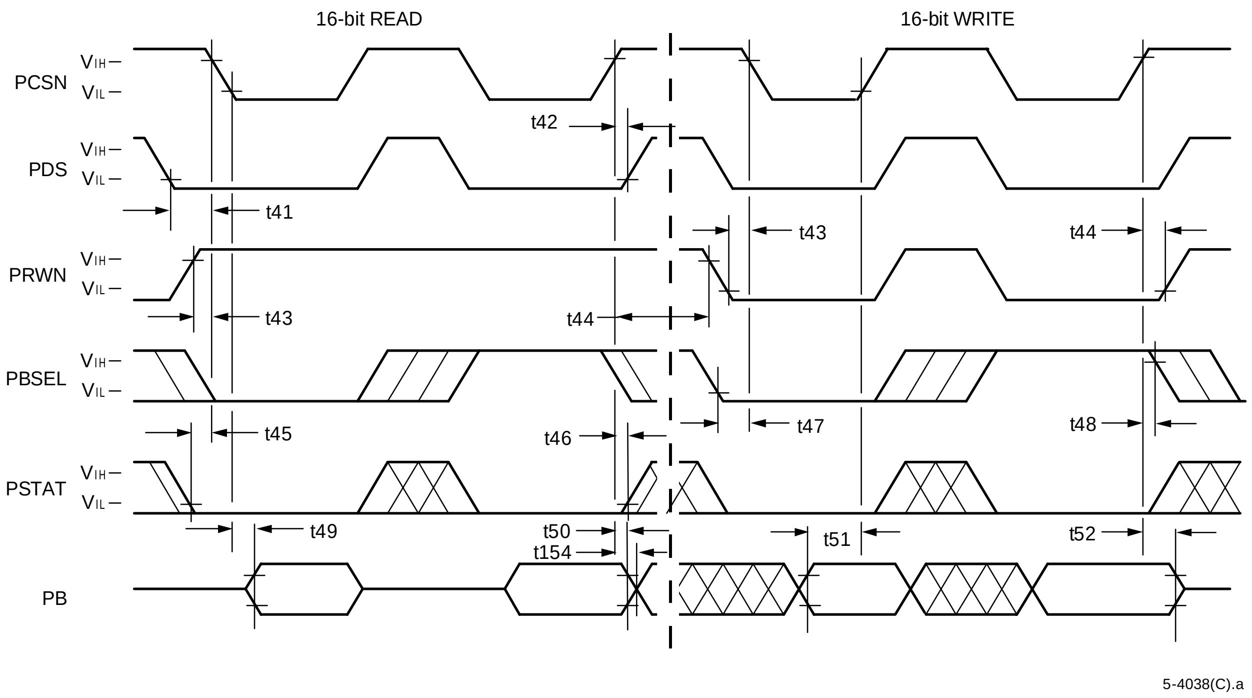


**Figure 32. PHIF16 *Motorola* Mode Signaling (Read and Write) Timing Diagram**

**Table 103. 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) | 4.5 | — | ns |
| t44 | PCSN to PRWN Hold (high to invalid) | 0 | — | ns |
| t45* | PSTAT to PCSN Setup (valid to low) | 4.5 | — | ns |
| t46* | PCSN to PSTAT Hold (high to invalid) | 0 | — | ns |
| t47* | PBSEL to PCSN Setup (valid to low) | 4.5 | — | ns |
| t48* | PCSN to PBSEL Hold (high to invalid) | 0 | — | ns |
| t51* | PB Write to PCSN Setup (valid to high) | 8 | — | 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 Figures 32 and 33. 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 104. Timing Characteristics for PHIF16 *Motorola* Mode Signaling (Read and Write)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t49* | PCSN to PB Read (low to valid) | — | 13 | ns |
| t50* | PCSN to PB Read Hold (high to invalid) | 0 | — | ns |
| t154 | PCSN to PB Read 3-state (high to 3-state) | — | 9 | ns |

* This timing diagram for the PHIF16 port shows accesses using the PCSN signal to initiate and complete a transaction. The transactions can also be initiated and completed with the PDS signal. An input/output transaction is initiated by PCSN or PDS going low, whichever comes last. For example, the timing requirement referenced to PCSN going low, t49, 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.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

Assuming an 8-bit external interface:

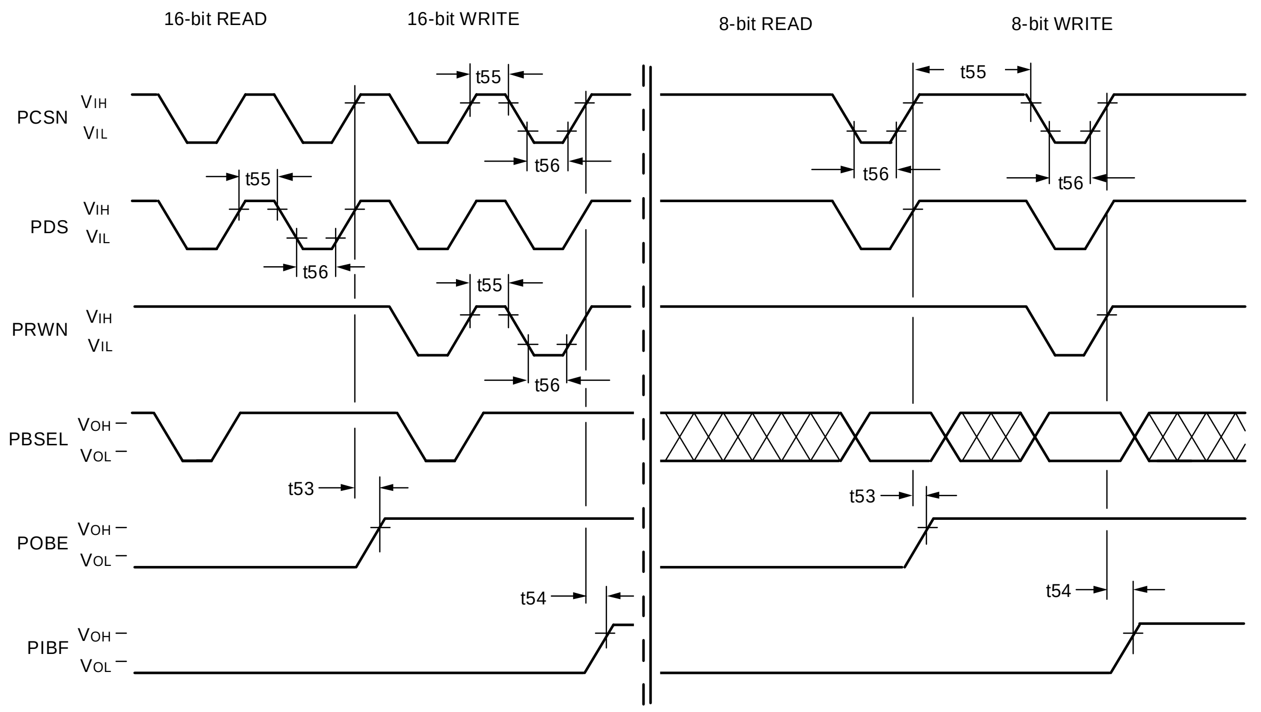


**Figure 33. PHIF16 *Motorola* Mode Signaling (Pulse Period and Flags) Timing Diagram**

**Table 105. 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) | 15 | — | ns |
| t56 | PCSN/PDS/PRWN Pulse Width (low to high) | 15 | — | ns |

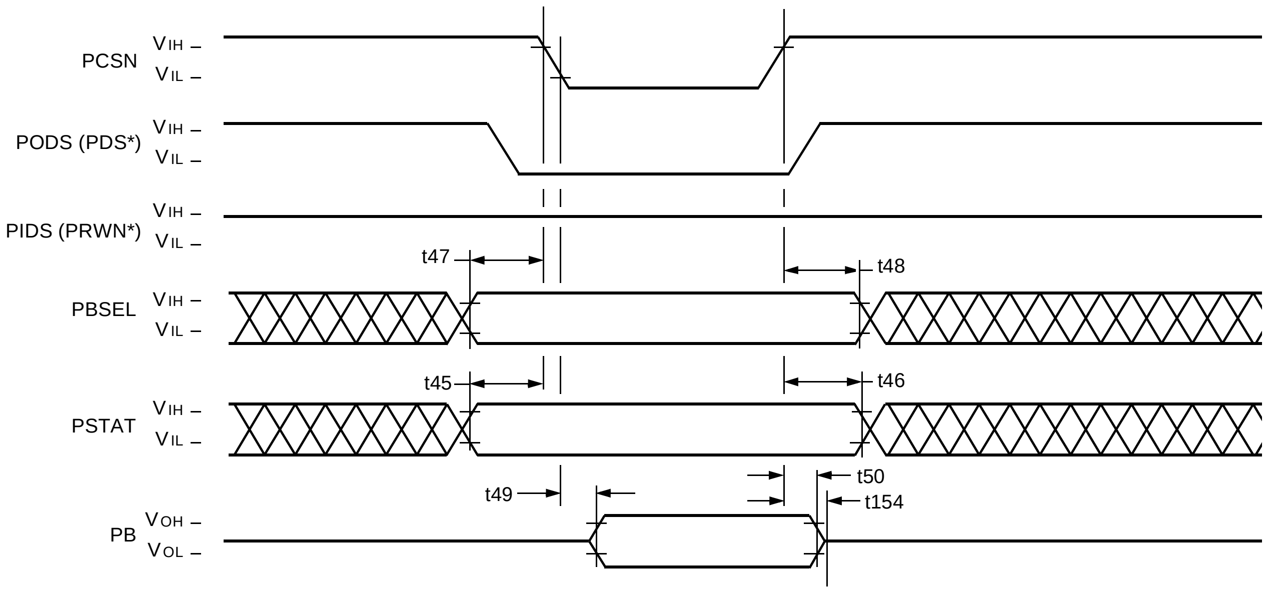
**Table 106. Timing Characteristics for PHIF16 *Motorola* Mode Signaling (Pulse Period and Flags)**

| 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 Figures 32 and 33. 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.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)



* *Motorola* mode signal name.

**Figure 34. PHIF16 *Intel* or *Motorola* Mode Signaling (Status Register Read) Timing Diagram**

**Table 107. 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) | 4.5 | — | ns |
| t46† | PCSN to PSTAT Hold (high to invalid) | 0 | — | ns |
| t47* | PBSEL to PCSN Setup (valid to low) | 4.5 | — | 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 108. 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) | — | 13 | ns |
| t50† | PCSN to PB Read Hold (high to invalid) | 0 | — | ns |
| t154† | PCSN to PB Read 3-state (high to 3-state) | — | 9 | 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.

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

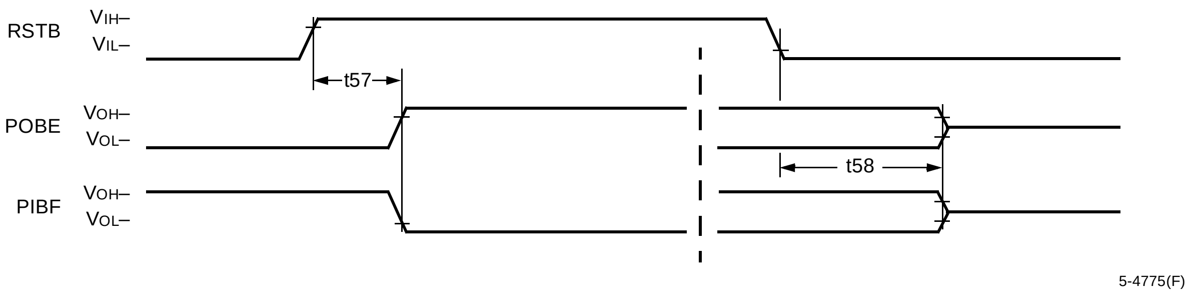
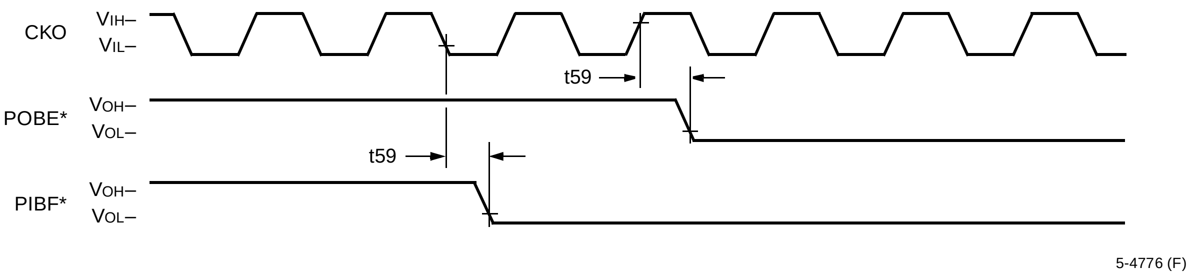


**Figure 35. PHIF16 PIBF and POBE Reset Timing Diagram**

**Table 109. PHIF16 Timing Characteristics for PIBF and POBE Reset**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t57 | RSTB Disable to POBE/PIBF* (high to valid) | — | 19 | ns |
| t58 | RSTB Enable to POBE/PIBF* (low to invalid) | 3 | 19 | 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.



* 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.

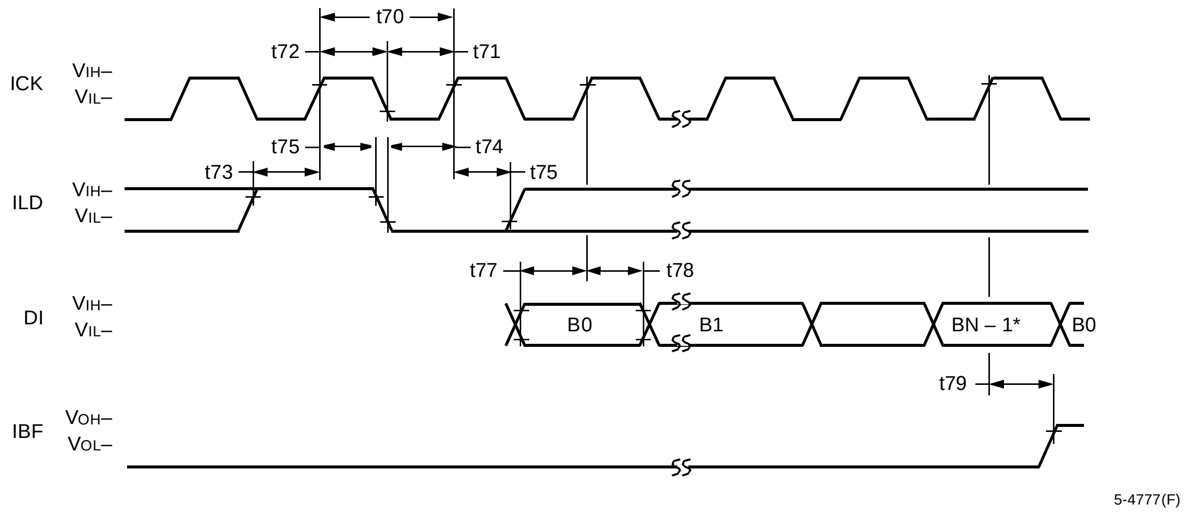
**Figure 36. PHIF16 PIBF and POBE Disable Timing Diagram**

**Table 110. PHIF16 Timing Characteristics for PIBF and POBE Disable**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t59 | CKO to POBE/PIBF Disable (high/low to disable) | — | 15 | ns |

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

### 10.9 Serial I/O Specifications



* N = 16 or 8 bits.

**Figure 37. SIO Passive Mode Input Timing Diagram**

**Table 111. Timing Requirements for Serial Inputs (Passive Mode)**

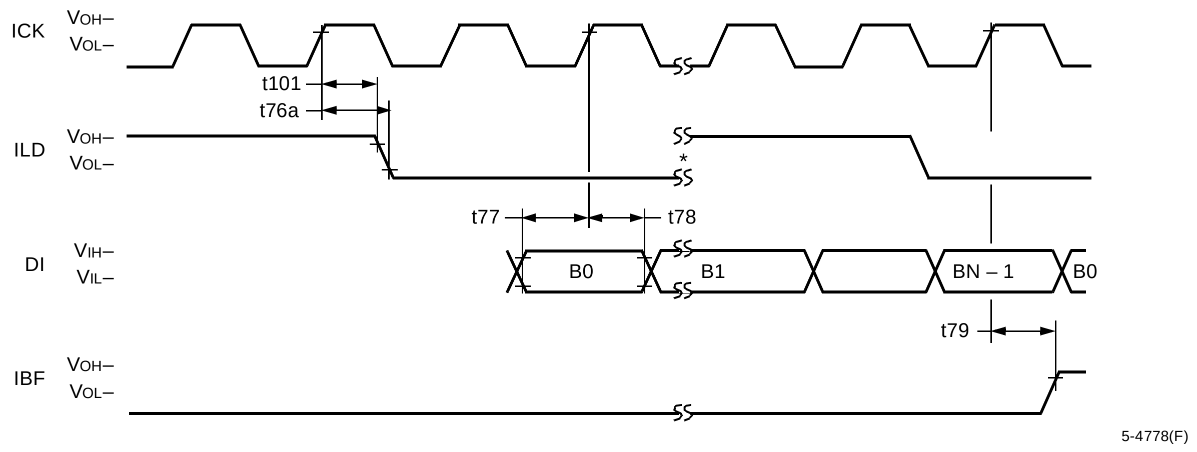
| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t70 | Clock Period (high to high)* | 40 | —† | ns |
| t71 | Clock Low Time (low to high) | 18 | — | ns |
| t72 | Clock High Time (high to low) | 18 | — | ns |
| t73 | Load High Setup (high to high) | 6 | — | ns |
| t74 | Load Low Setup (low to high) | 6 | — | ns |
| t75 | Load High Hold (high to invalid) | 0 | — | ns |
| t77 | Data Setup (valid to high) | 6 | — | ns |
| t78 | Data Hold (high to invalid) | 0 | — | ns |

* For multiprocessor mode, see note in Section 10.10.
† Device is fully static; t70 is tested at 200 ns.

**Table 112. Timing Characteristics for Serial Outputs (Passive Mode)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t79 | IBF Delay (high to high) | — | 22 | ns |

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)



* ILD goes high during bit 6 (of 0:15), N = 8 or 16.

**Figure 38. SIO Active Mode Input Timing Diagram**

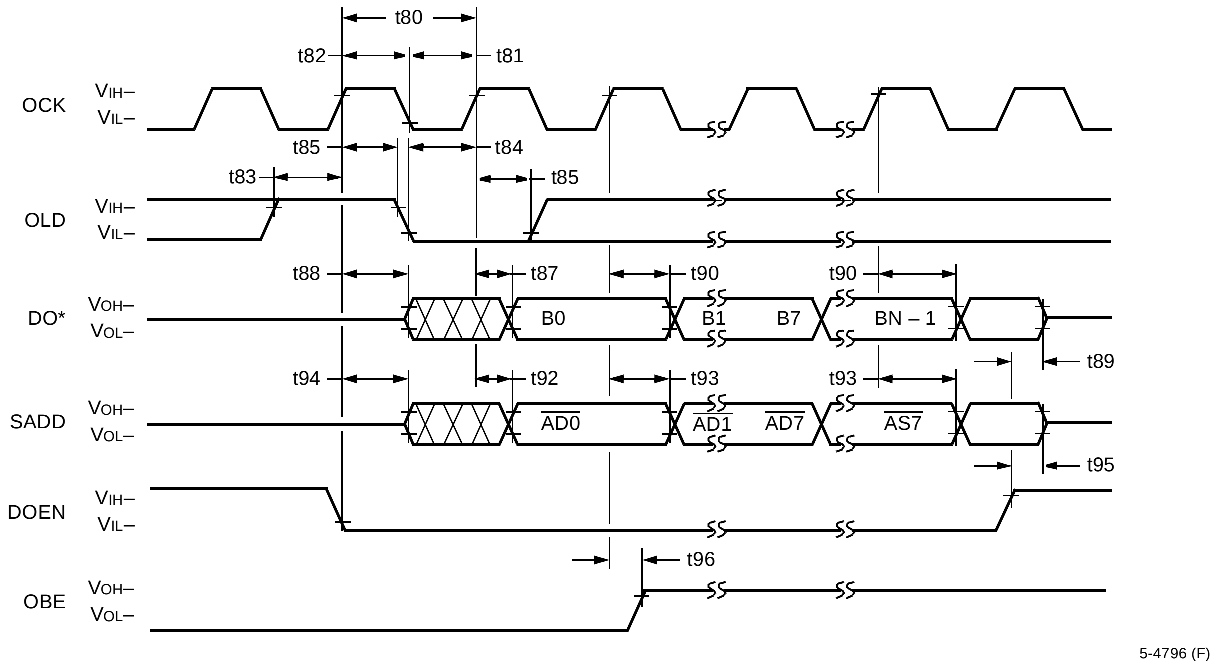
**Table 113. Timing Requirements for Serial Inputs (Active Mode)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t77 | Data Setup (valid to high) | 6 | — | ns |
| t78 | Data Hold (high to invalid) | 0 | — | ns |

**Table 114. Timing Characteristics for Serial Outputs (Active Mode)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t76a | ILD Delay (high to low) | — | 22 | ns |
| t101 | ILD Hold (high to invalid) | 2 | — | ns |
| t79 | IBF Delay (high to high) | — | 22 | ns |

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)



* See **sioc** register, MSB1 field to determine if B0 is the MSB or LSB. See **sioc** register, ILEN1 field to determine if the DO word length is 8 bits or 16 bits.

**Figure 39. SIO Passive Mode Output Timing Diagram**

**Table 115. Timing Requirements for Serial Inputs (Passive Mode)**

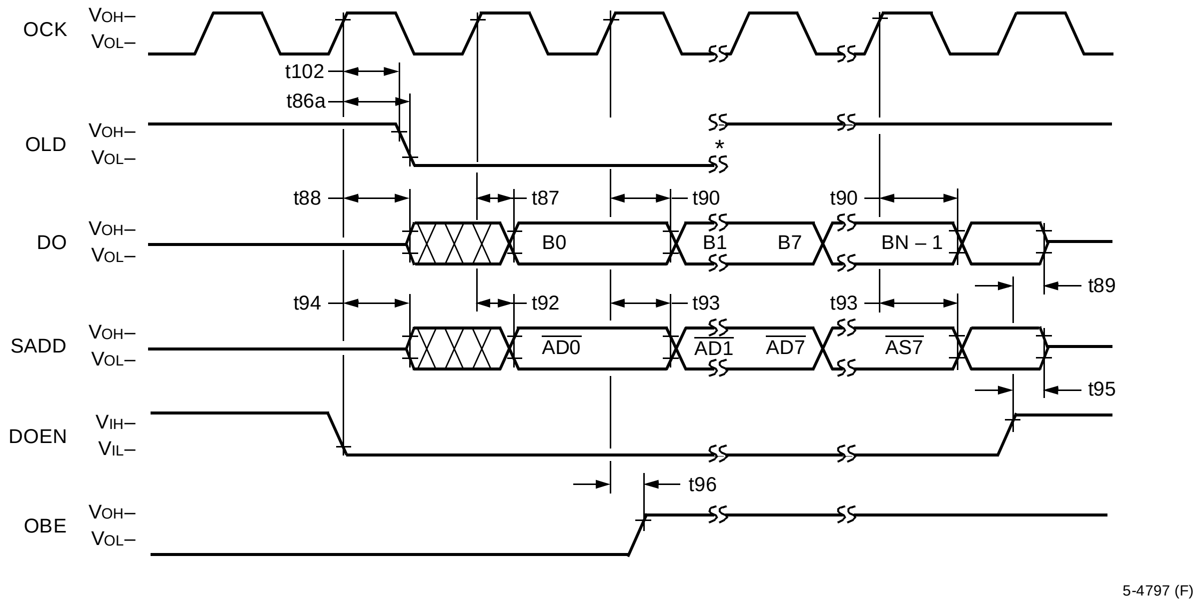
| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t80 | Clock Period (high to high)* | 40 | —† | ns |
| t81 | Clock Low Time (low to high) | 18 | — | ns |
| t82 | Clock High Time (high to low) | 18 | — | ns |
| t83 | Load High Setup (high to high) | 6 | — | ns |
| t84 | Load Low Setup (low to high) | 6 | — | ns |
| t85 | Load Hold (high to invalid) | 0 | — | ns |

* For multiprocessor mode, see note in Section 10.10.
† Device is fully static; t80 is tested at 200 ns.

**Table 116. Timing Characteristics for Serial Outputs (Passive Mode)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t87 | Data Delay (high to valid) | — | 22 | ns |
| t88 | Enable Data Delay (low to active) | — | 22 | ns |
| t89 | Disable Data Delay (high to 3-state) | — | 22 | ns |
| t90 | Data Hold (high to invalid) | 2 | — | ns |
| t92 | Address Delay (high to valid) | — | 22 | ns |
| t93 | Address Hold (high to invalid) | 2 | — | ns |
| t94 | Enable Delay (low to active) | — | 22 | ns |
| t95 | Disable Delay (high to 3-state) | — | 22 | ns |
| t96 | OBE Delay (high to high) | — | 22 | ns |

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)



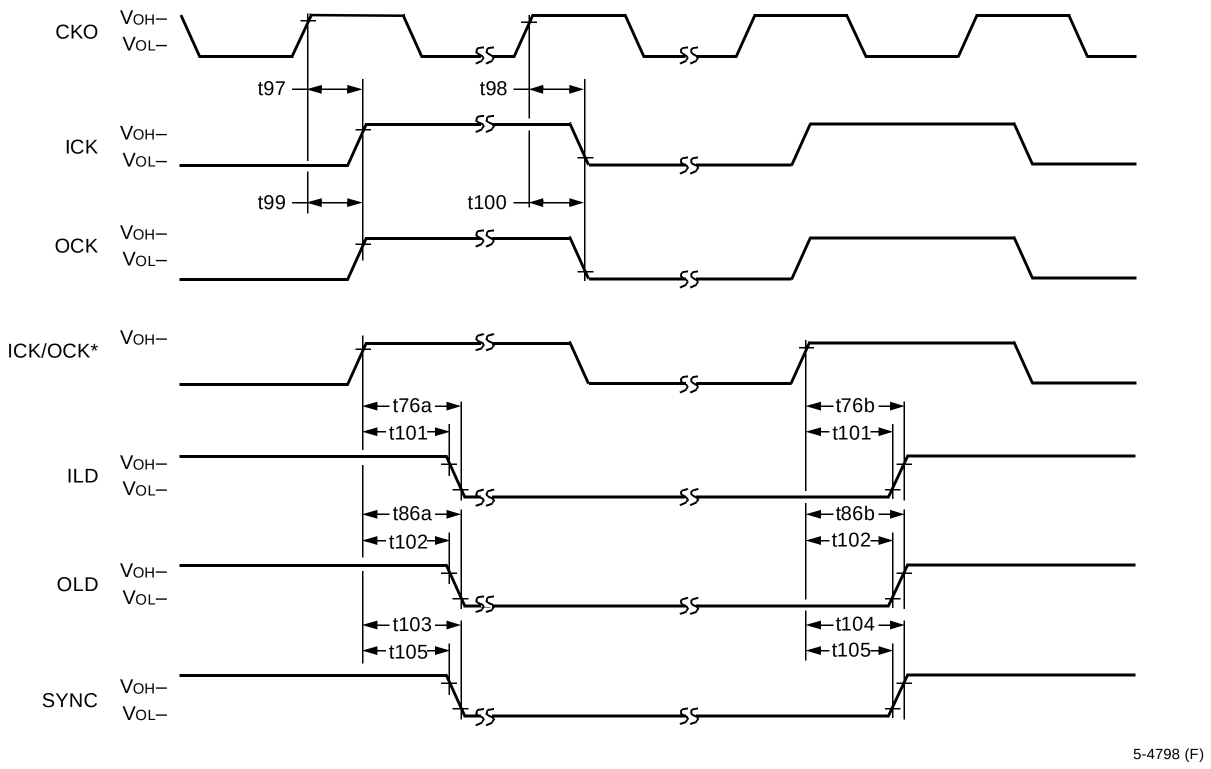
* OLD goes high at the end of bit 6 of 0:15.

**Figure 40. SIO Active Mode Output Timing Diagram**

**Table 117. Timing Characteristics for Serial Outputs**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t86a | OLD Delay (high to low) | — | 22 | ns |
| t102 | OLD Hold (high to invalid) | 2 | — | ns |
| t87 | Data Delay (high to valid) | — | 22 | ns |
| t88 | Enable Data Delay (low to active) | — | 22 | ns |
| t89 | Disable Data Delay (high to 3-state) | — | 22 | ns |
| t90 | Data Hold (high to invalid) | 2 | — | ns |
| t92 | Address Delay (high to valid) | — | 22 | ns |
| t93 | Address Hold (high to invalid) | 2 | — | ns |
| t94 | Enable Delay (low to active) | — | 22 | ns |
| t95 | Disable Delay (high to 3-state) | — | 22 | ns |
| t96 | OBE Delay (high to high) | — | 22 | ns |

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)



* See **sioc** register, LD1 field.

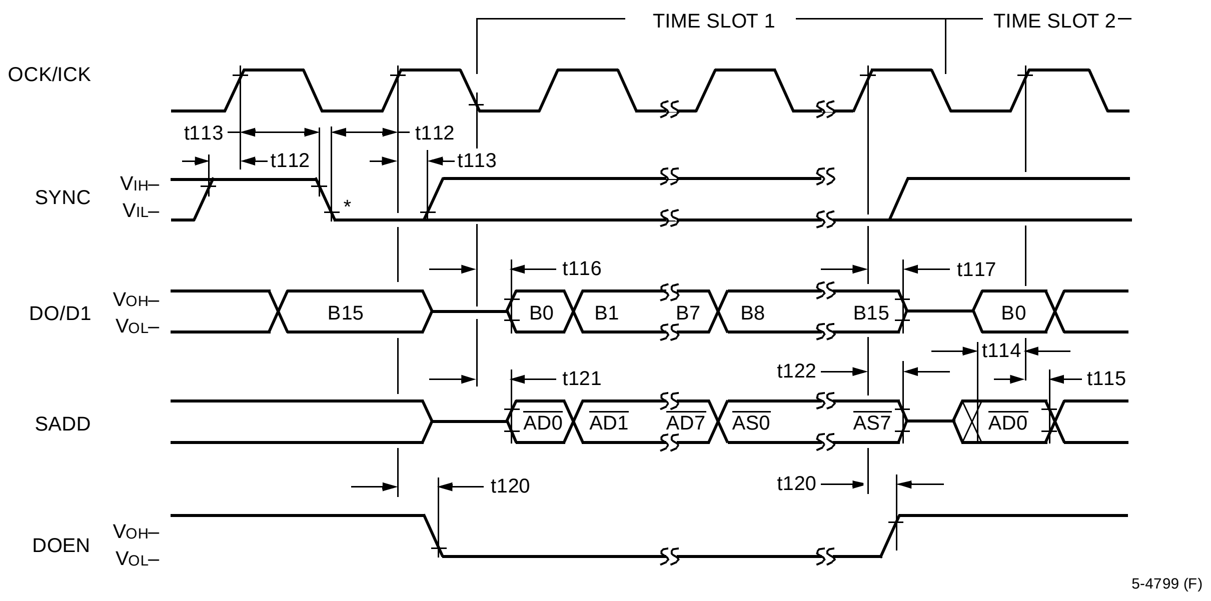
**Figure 41. Serial I/O Active Mode Clock Timing**

**Table 118. Timing Characteristics for Signal Generation**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t97 | ICK Delay (high to high) | — | 18 | ns |
| t98 | ICK Delay (high to low) | — | 18 | ns |
| t99 | OCK Delay (high to high) | — | 18 | ns |
| t100 | OCK Delay (high to low) | — | 18 | ns |
| t76a | ILD Delay (high to low) | — | 22 | ns |
| t76b | ILD Delay (high to high) | — | 22 | ns |
| t101 | ILD Hold (high to invalid) | 2 | — | ns |
| t86a | OLD Delay (high to low) | — | 22 | ns |
| t86b | OLD Delay (high to high) | — | 22 | ns |
| t102 | OLD Hold (high to invalid) | 2 | — | ns |
| t103 | SYNC Delay (high to low) | — | 22 | ns |
| t104 | SYNC Delay (high to high) | — | 22 | ns |
| t105 | SYNC Hold (high to invalid) | 2 | — | ns |

## 10 Timing Characteristics and Requirements for 5 V Operation (continued)

### 10.10 Multiprocessor Communication



* Negative edge initiates time slot 0.

**Figure 42. SIO Multiprocessor Timing Diagram**

**Note:** All serial I/O timing requirements and characteristics still apply except the minimum clock period in passive multiprocessor mode, assuming 50% duty cycle, is calculated as (t77 + t116) x 2.

**Table 119. Timing Requirements for SIO Multiprocessor Communication**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t112 | Sync Setup (high/low to high) | 22 | — | ns |
| t113 | Sync Hold (high to high/low) | 0 | — | ns |
| t114 | Address Setup (valid to high) | 9 | — | ns |
| t115 | Address Hold (high to invalid) | 0 | — | ns |

**Table 120. Timing Characteristics for SIO Multiprocessor Communication**

| Abbreviated Reference* | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t116 | Data Delay (bit 0 only) (low to valid) | — | 22 | ns |
| t117 | Data Disable Delay (high to 3-state) | — | 20 | ns |
| t120 | DOEN Valid Delay (high to valid) | — | 16 | ns |
| t121 | Address Delay (bit 0 only) (low to valid) | — | 22 | ns |
| t122 | Address Disable Delay (high to 3-state) | — | 20 | ns |

* With capacitance load on ICK, OCK, DO, SYNC, and SADD = 100 pF, add 4 ns to t116—t122.

## 11 Timing Characteristics and Requirements for 3 V Operation

The following timing characteristics and requirements are preliminary information and are subject to change. 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 Section 8.3.)

$V_{DD}$ = 3.3 V ± 0.3 V, $V_{SS}$ = 0 V (See Section 8.3.)

Capacitance load on outputs ($C_L$) = 50 pF

Output characteristics can be derated as a function of load capacitance (CL).

| | |
|---|---|
| All outputs except CKO: | 0.03 ns/pF ≤ $dt/dC_L$ ≤ 0.07 ns/pF for 10 ≤ $C_L$ ≤ 100 pF |
| CKO: | 0.06 ns/pF ≤ $dt/dC_L$ ≤ 0.12 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 is 30 pF instead of 50 pF, the derating for a rising edge is (30 – 50) pF x 0.06 ns/pF = 1.2 ns **less** than the specified rise time or delay that includes a rise time.

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):

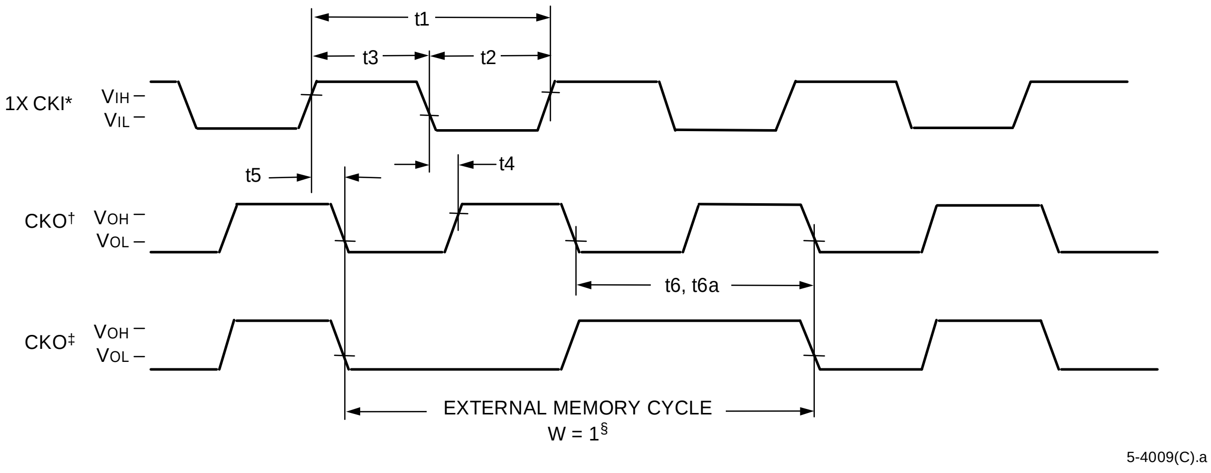
- $C_{LOAD}$ = 50 pF
- Timing reference levels for delays = $V_{IH}$, $V_{IL}$
- 3-state delays measured to the high-impedance state of the output driver

For the timing diagrams, see Table 76 for input clock requirements.

Unless otherwise noted, CKO in the timing diagrams is the free-running CKO.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

### 11.1 DSP Clock Generation



* See Table 76 for input clock electrical requirements.
† Free-running clock.
‡ Wait-stated clock (see Table 41).
§ W = number of wait-states.

**Figure 43. I/O Clock Timing Diagram**

**Table 121. 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 for 1X input clock option, and memory hold time is tested at 0.1 s.

**Table 122. Timing Characteristics for Input Clock and Output Clock**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t4 | Clock Out High Delay | — | 14 | ns |
| t5 | Clock Out Low Delay (high to low) | — | 14 | ns |
| t6 | Clock Out Period (low to low) | T* | — | ns |
| t6a | Clock Out Period with SLOWCKI Bit Set in **powerc** Register (low to low) | 0.74 | 3.8 | µs |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

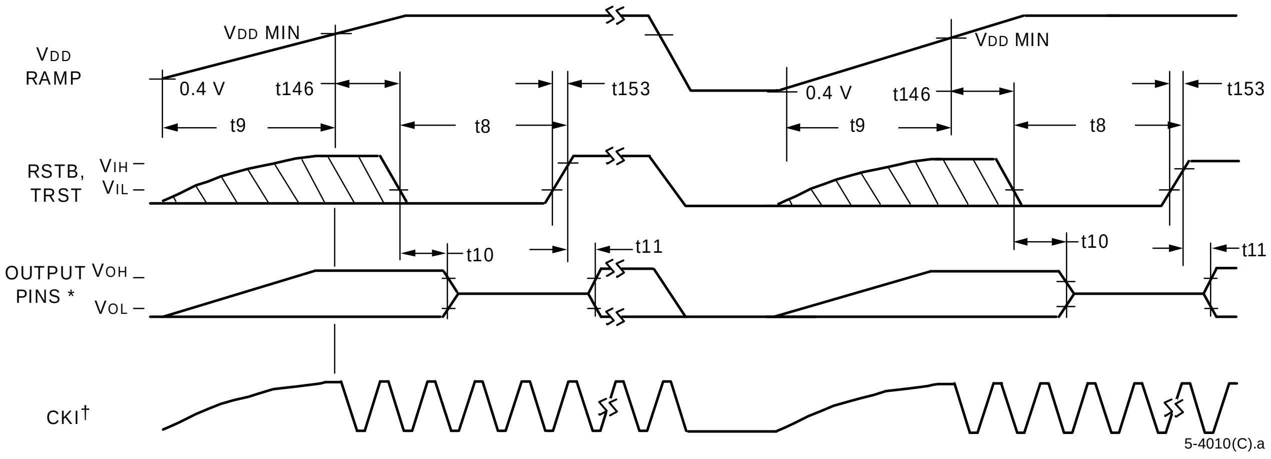
### 11.2 Reset Circuit

The DSP1620 has two external reset pins: RSTB and TRST. At initial powerup, or if the supply voltage falls below VDD MIN* and a device reset is required, both TRST and RSTB must be asserted to initialize the device. Figure 44 shows two separate events:

1. Chip reset at initial powerup.
2. Chip 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.**

* See Table 74, Recommended Operating Conditions.



* When the level of RSTB = 0 and INT0 = 1, all output and bidirectional pins (except TDO, which 3-states by JTAG control) are put in a 3-state condition. When the level of RSTB = 0 and INT0 = 0, EROM, ERAMHI, ERAMLO, IO, and RWN outputs remain high, and CKO remains a free-running clock.
† See Table 76 for input clock electrical requirements.

**Figure 44. Powerup Reset and Chip Reset Timing Diagram**

**Table 123. Timing Requirements for Powerup Reset and Chip 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, set by CKI or by CKI and the PLL parameters.

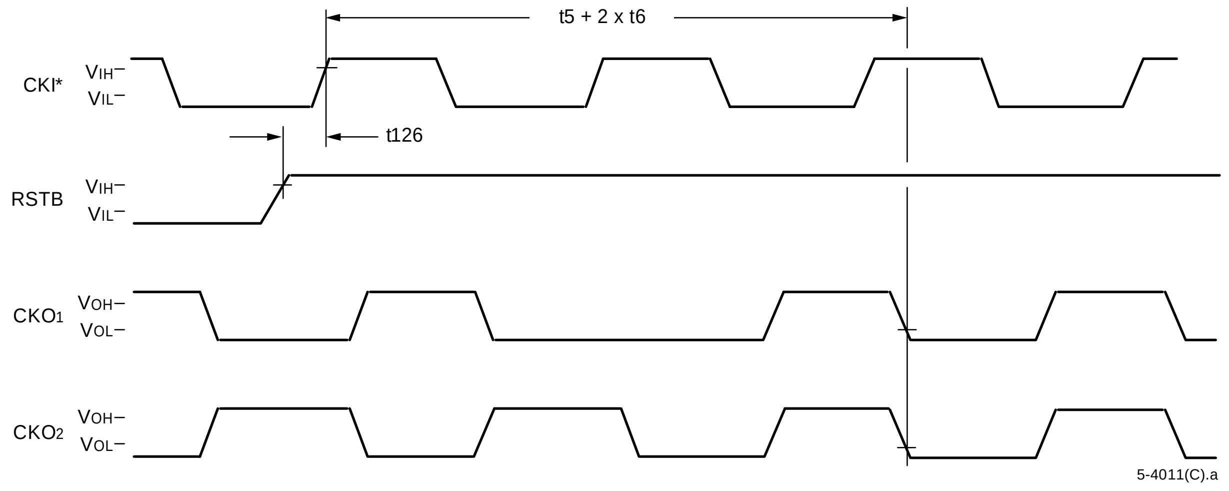
**Table 124. Timing Characteristics for Powerup Reset and Chip Reset**

| 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.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

### 11.3 Reset Synchronization



* See Table 76 for input clock electrical requirements.

Notes:

CKO1 and CKO2 are two possible CKO states before reset. CKO is free-running.

If the rising edge of RSTB (low to high) is captured instead by the falling edge of CKO (high to low), CKO and CKI will be in-phase at t5 + 2 x t6.

**Figure 45. Reset Synchronization Timing**

**Table 125. 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, set by CKI or by CKI and the PLL parameters.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

### 11.4 JTAG I/O Specifications

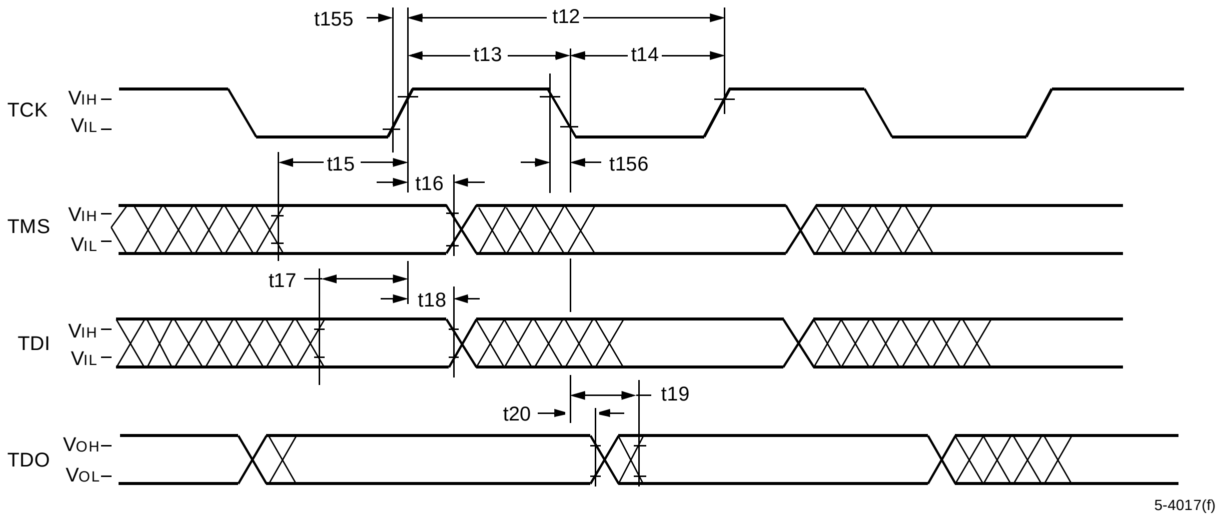


**Figure 46. JTAG Timing Diagram**

**Table 126. Timing Requirements for JTAG Input/Output**

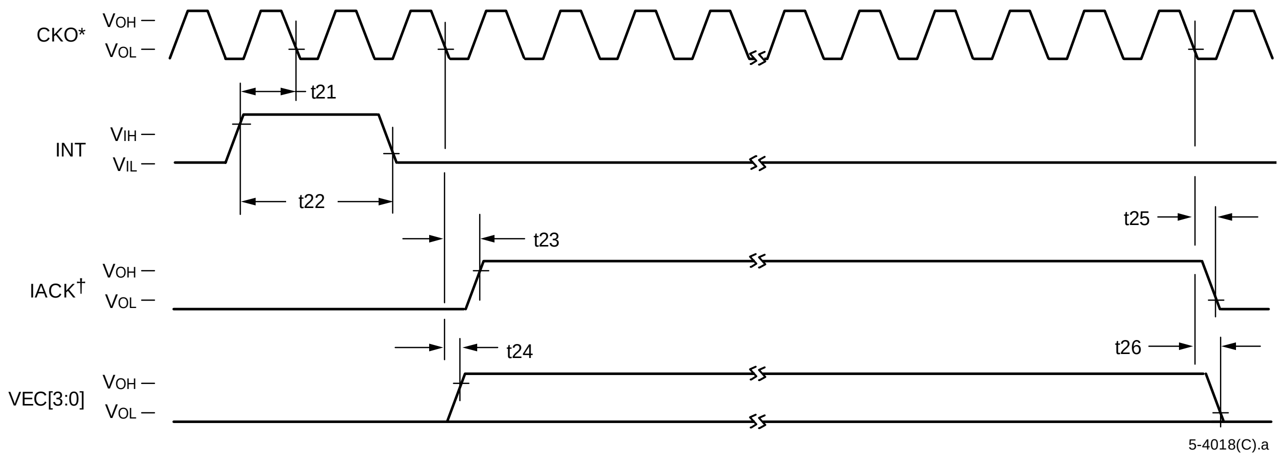
| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t12 | TCK Period (high to high) | 50 | — | ns |
| t13 | TCK High Time (high to low) | 22.5 | — | ns |
| t14 | TCK Low Time (low to high) | 22.5 | — | ns |
| t155 | TCK Rise Transition Time (low to high) | 0.6 | — | V/ns |
| 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) | 2 | — | ns |
| t17 | TDI Setup Time (valid to high) | 7.5 | — | ns |
| t18 | TDI Hold Time (high to invalid) | 2 | — | ns |

**Table 127. Timing Characteristics for JTAG Input/Output**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t19 | TDO Delay (low to valid) | — | 19 | ns |
| t20 | TDO Hold (low to invalid) | 0 | — | ns |

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

### 11.5 Interrupt



* CKO is free-running.
† IACK assertion is guaranteed to be enclosed by VEC[3:0] assertion.

**Figure 47. Interrupt Timing Diagram**

**Table 128. Timing Requirements for Interrupt**

**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) | 19 | — | ns |
| t22 | INT Assertion Time (high to low) | 2T* | — | ns |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

**Table 129. Timing Characteristics for Interrupt**

**Note:** Interrupt is asserted during an interruptible instruction and no other pending interrupts.

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t23 | IACK Assertion Time (low to high) | — | T/2 + 10* | ns |
| t24 | VEC Assertion Time (low to high) | — | 12.5 | ns |
| t25 | IACK Invalid Time (low to low) | — | 10 | ns |
| t26 | VEC Invalid Time (low to low) | — | 12.5 | ns |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

### 11.6 Bit Input/Output (BIO)

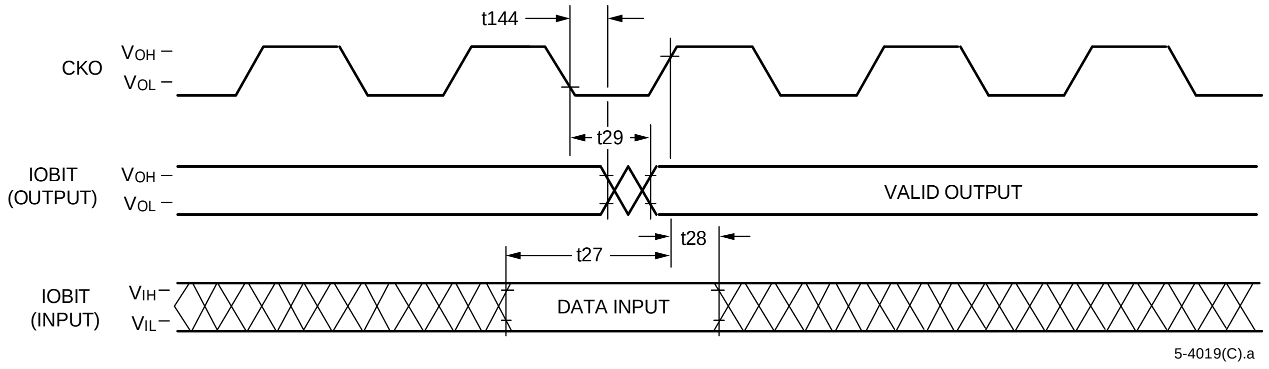


**Figure 48. Write Outputs Followed by Read Inputs (cbit = Immediate; a1 = sbit)**

**Table 130. Timing Requirements for BIO Input Read**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t27 | IOBIT Input Setup Time (valid to high) | 15 | — | ns |
| t28 | IOBIT Input Hold Time (high to invalid) | 0 | — | ns |

**Table 131. Timing Characteristics for BIO Output**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t29 | IOBIT Output Valid Time (low to valid) | — | 9 | ns |
| t144 | IOBIT Output Hold Time (low to invalid) | 1 | — | ns |

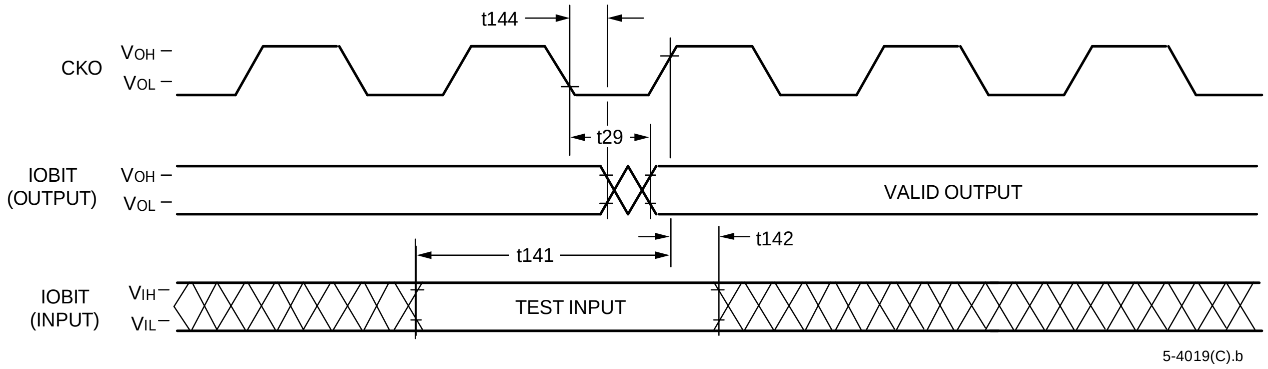


**Figure 49. Write Outputs and Test Inputs (cbit = Immediate)**

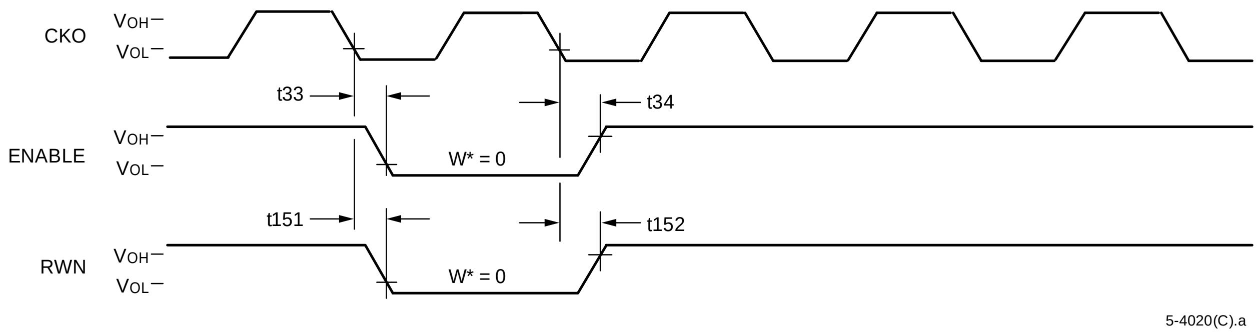
**Table 132. Timing Requirements for BIO Input Test**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t141 | IOBIT Input Setup Time (valid to low) | 15 | — | ns |
| t142 | IOBIT Input Hold Time (low to invalid) | 0 | — | ns |

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

### 11.7 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 *DSP1620 Digital Signal Processor* Information Manual for a detailed description of the external memory interface including other functional diagrams. The term ENABLE refers to EROM, ERAMX, IO, ERAMHI, and ERAMLO.



* W = number of wait-states.

**Figure 50. Enable Transition Timing**

**Table 133. Enable Transition Timing When Memory Enables and RWN Are Not Delayed (When ioc DENB[4:0] = 0x00 and ioc RWNADV = 0x1)**

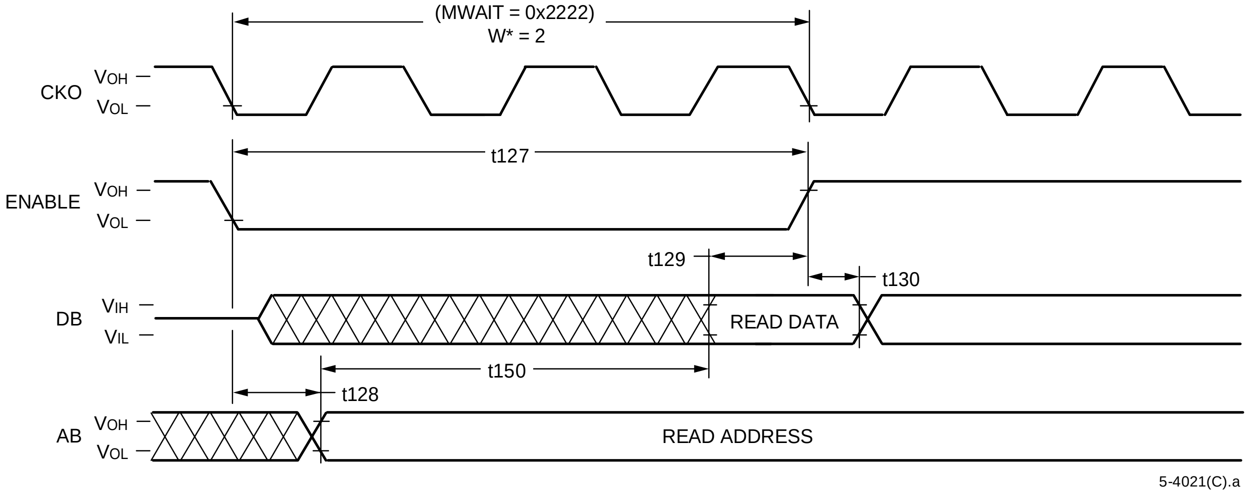
| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t33 | CKO to ENABLE Active (low to low) | 0 | 4.5 | ns |
| t34 | CKO to ENABLE Inactive (low to high) | –1 | 4 | ns |
| t151 | CKO to RWN (RWNADV = 0x1) Active (low to low) | 0 | 4.5 | ns |
| t152 | CKO to RWN (RWNADV = 0x1) Inactive (low to high) | –1 | 4 | ns |

**Table 134. Timing Characteristics for Delayed External Memory Enables (When ioc DENB[4:0] = 0x1F)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t33 | CKO to Delayed ENABLE Active (low to low) | T/2* | T/2 + 7* | ns |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)



* W = number of wait-states.

**Figure 51. External Memory Data Read Timing Diagram**

**Table 135. Timing Characteristics for External Memory Access**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t127 | Enable Width (low to high) | T(1 + W) – 2* | — | ns |
| t128 | Address Valid (enable low to valid) | — | 2 | ns |

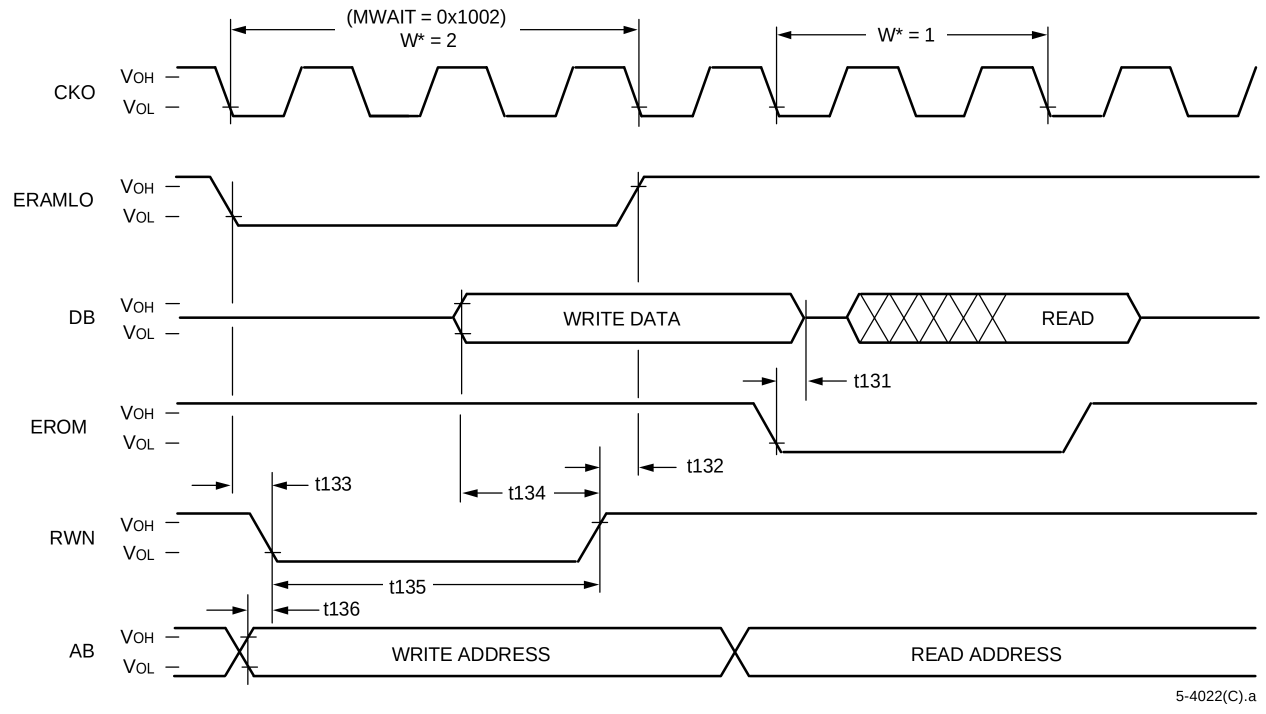
* T = internal clock period, set by CKI or by CKI and the PLL parameters.

**Table 136. Timing Requirements for External Memory Read (EROM, ERAMHI, I/O, ERAMLO)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t129 | Read Data Setup (valid to enable high) | 13 | — | ns |
| t130 | Read Data Hold (enable high to hold) | 0 | — | ns |
| t150 | External Memory Access Time (valid to valid) | — | T(1 + W) – 15* | ns |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)



* W = number of wait-states.

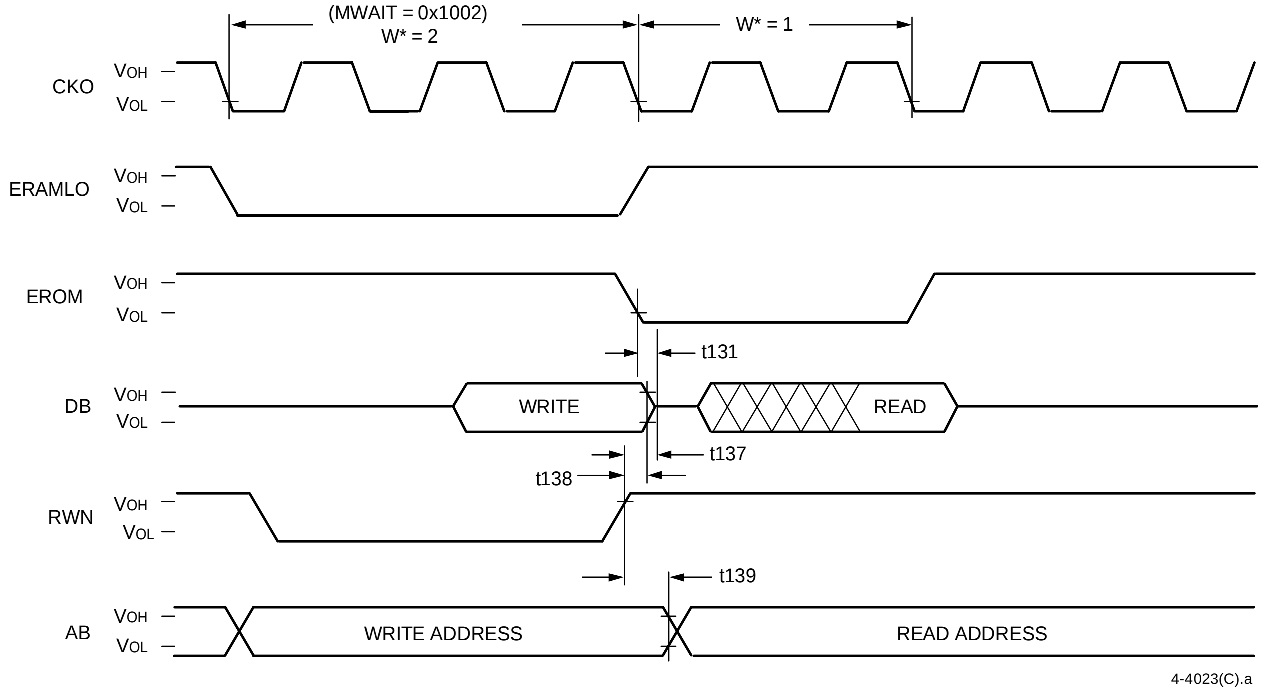
**Figure 52. External Memory Data Write Timing Diagram**

**Table 137. Timing Characteristics for External Memory Data Write (All Enables)***

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t131 | Write Overlap (enable low to 3-state) | — | 0 | ns |
| t132 | RWN Advance (RWN high to enable high) | 0 | 3 | ns |
| t133 | RWN Delay (enable low to RWN low) | 0 | 3 | ns |
| t134 | Write Data Setup (data valid to RWN high) | T(1 + W)/2 – 4† | — | ns |
| t135 | RWN Width (low to high) | T(1 + W) – 4† | — | ns |
| t136 | Write Address Setup (address valid to RWN low) | 0 | — | ns |

* **ioc** RWNADV = 0x0.
† T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)



* W = number of wait-states.

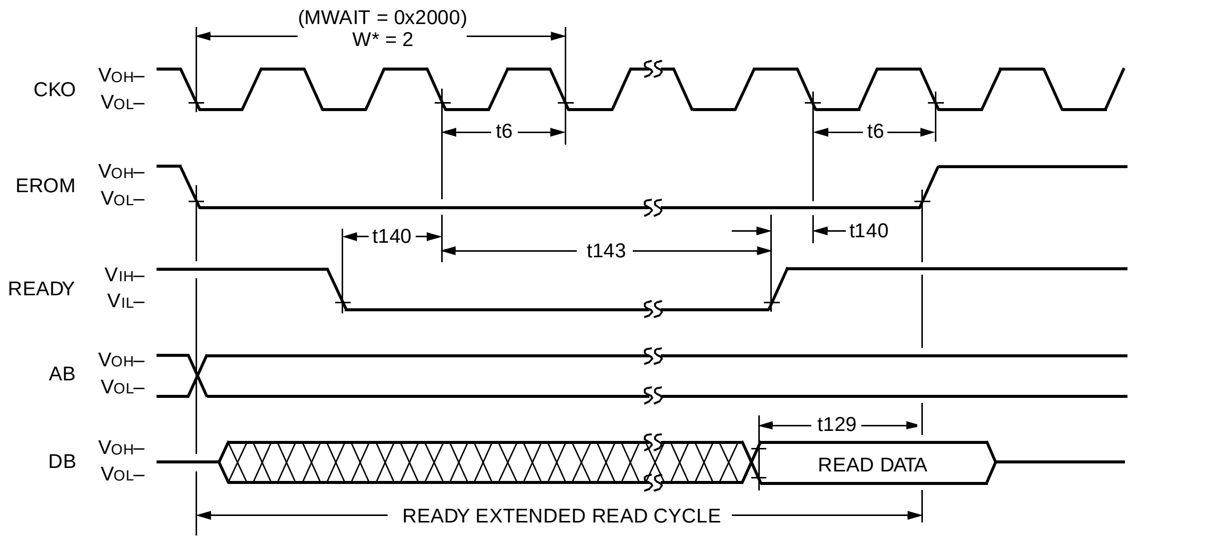
**Figure 53. Write Cycle Followed by Read Cycle**

**Table 138. Timing Characteristics for Write Cycle Followed by Read Cycle***

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t131 | Write Overlap (enable low to 3-state) | — | 0 | ns |
| t137 | Write Data 3-state (RWN high to 3-state) | — | 2 | ns |
| t138 | Write Data Hold (RWN high to data hold) | 0 | — | ns |
| t139 | Write Address Hold (RWN high to address hold) | 0 | — | ns |

* **ioc** RWNADV = 0x0.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)



* W = number of wait-states.

**Figure 54. READY Extended Read Cycle Timing**

**Table 139. Timing Requirements for READY Extended Read Cycle Timing**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t6 | Clock Out Period (low to low) | T* | — | ns |
| t129 | Read Data Setup (valid to enable high) | 13 | — | ns |
| t140 | READY Setup (valid to CKO low) | –1 | — | ns |
| t143 | READY Hold (CKO low to invalid) | 8 | — | ns |

* T = internal clock period, set by CKI or by CKI and the PLL parameters.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

### 11.8 PHIF16 Specifications

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.

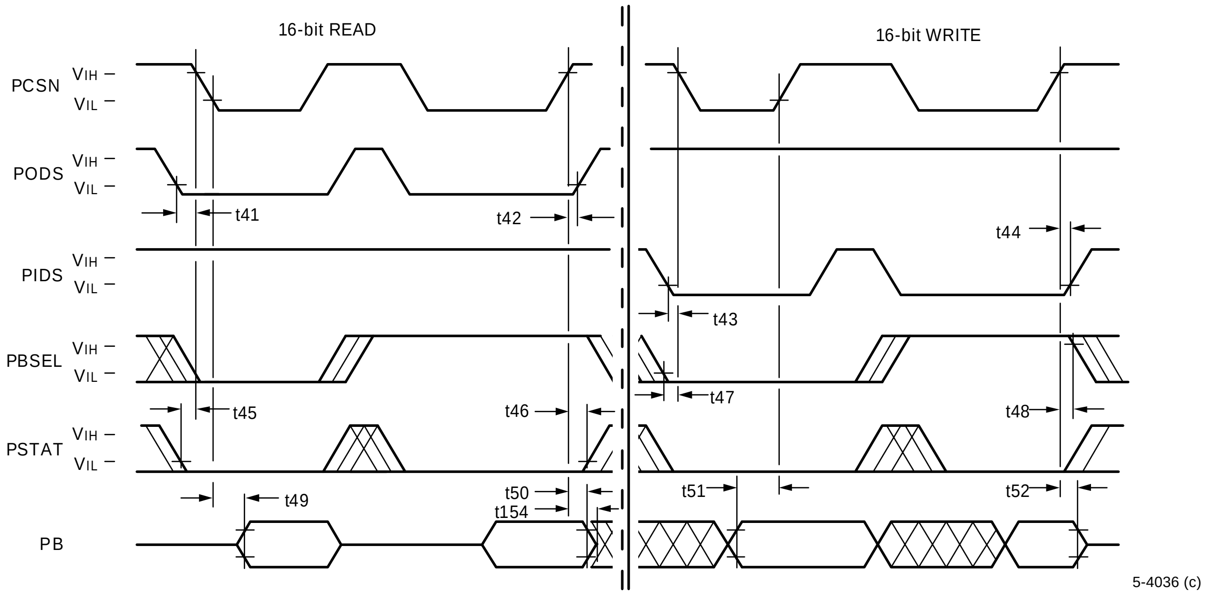


**Figure 55. PHIF16 *Intel* Mode Signaling (Read and Write) Timing Diagram**

**Table 140. 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 |

* This timing diagram for the PHIF16 port 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. For example, the timing requirement referenced to PCSN going low, t45, should be referenced to PODS going low, if PODS goes low after PCSN. 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. All requirements referenced to PCSN apply to PIDS or PODS, if PIDS or PODS is the controlling signal.

**Table 141. 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) | 0 | — | ns |
| t154 | PCSN to PB Read 3-state (high to 3-state) | — | 8 | ns |

* This timing diagram for the PHIF16 port 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. For example, the timing requirement referenced to PCSN going low, t49, should be referenced to PODS going low, if PODS goes low after PCSN. 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. All requirements referenced to PCSN apply to PIDS or PODS, if PIDS or PODS is the controlling signal.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

Assuming an 8-bit external interface:

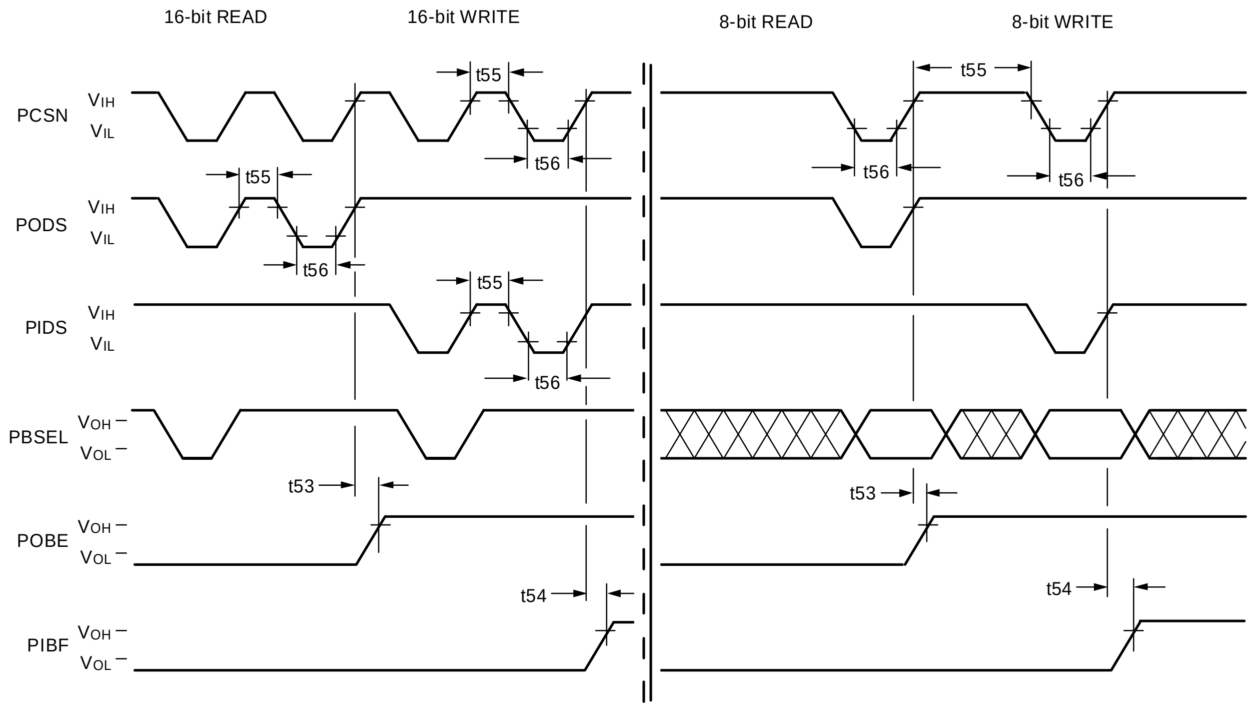


**Figure 56. PHIF16 *Intel* Mode Signaling (Pulse Period and Flags) Timing Diagram**

**Table 142. 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 143. 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) | — | 17 | ns |
| t54* | PCSN/PIDS to PIBF†(high to high) | — | 17 | 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.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

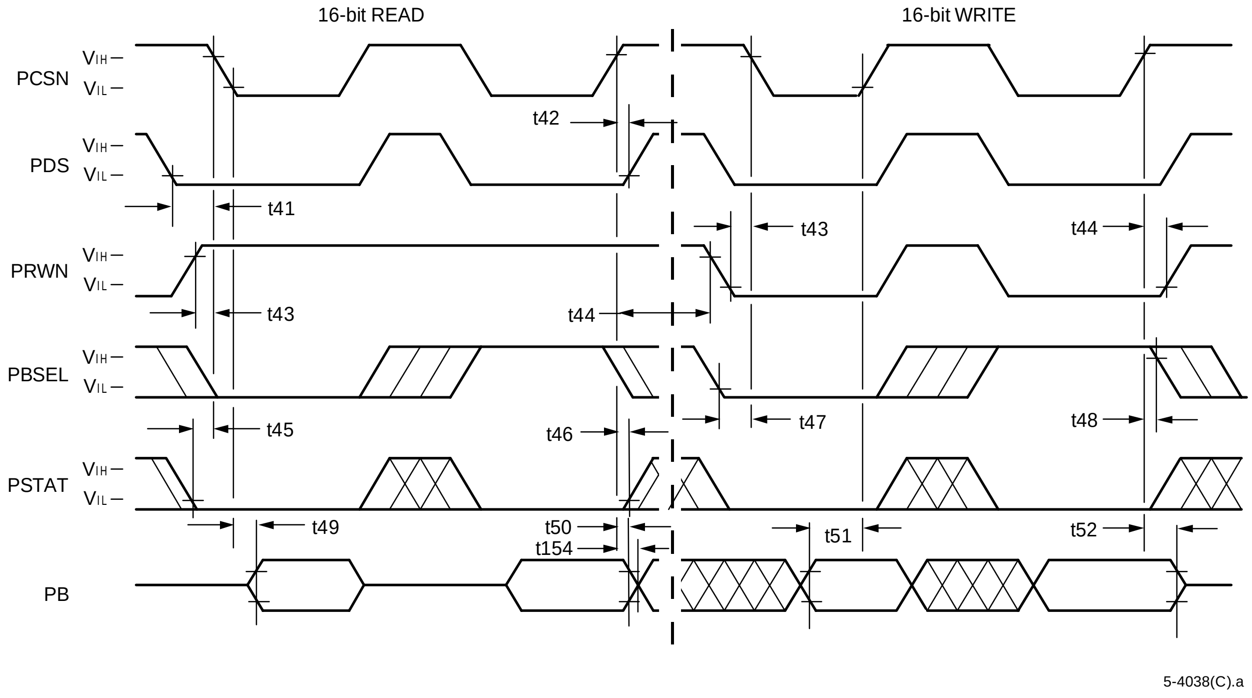


**Figure 57. PHIF16 *Motorola* Mode Signaling (Read and Write) Timing Diagram**

**Table 144. 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 Figures 57 and 58. 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 145. 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 Hold (high to invalid) | 0 | — | ns |
| t154 | PCSN to PB Read 3-state (high to 3-state) | — | 8 | ns |

* This timing diagram for the PHIF16 port shows accesses using the PCSN signal to initiate and complete a transaction. The transactions can also be initiated and completed with the PDS signal. An input/output transaction is initiated by PCSN or PDS going low, whichever comes last. For example, the timing requirement referenced to PCSN going low, t49, 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.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

Assuming an 8-bit external interface:

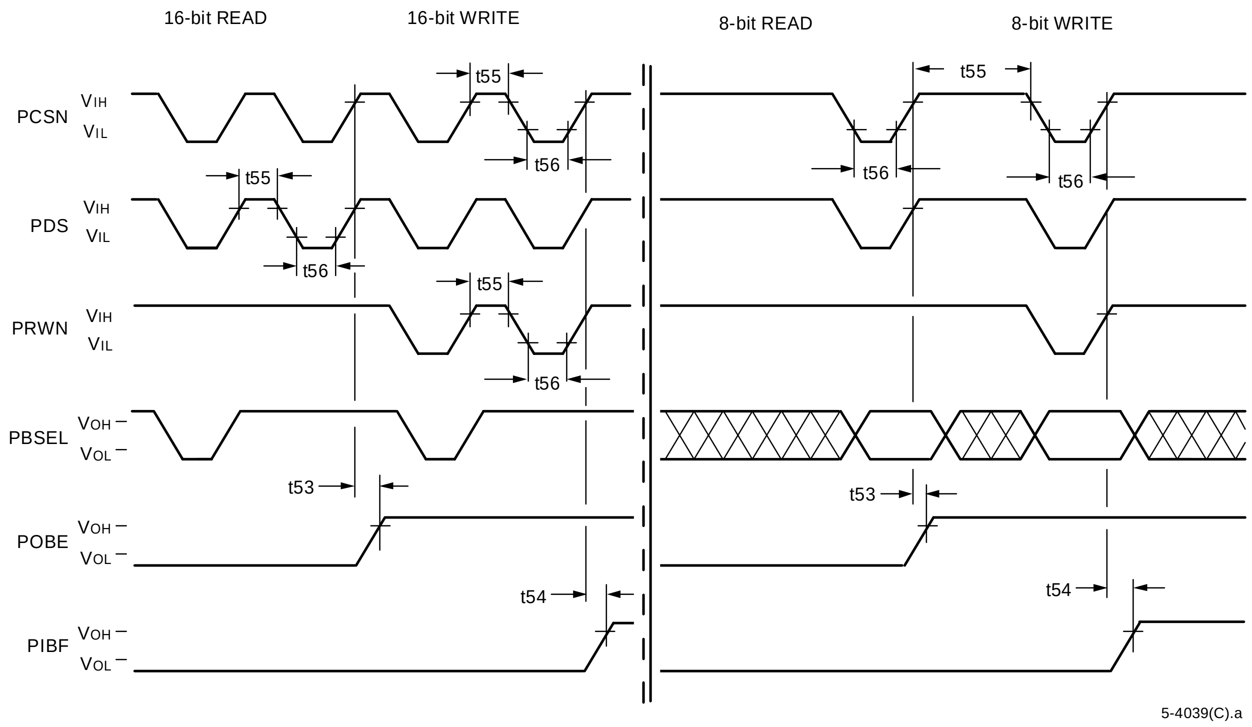


**Figure 58. PHIF16 *Motorola* Mode Signaling (Pulse Period and Flags) Timing Diagram**

**Table 146. 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 |

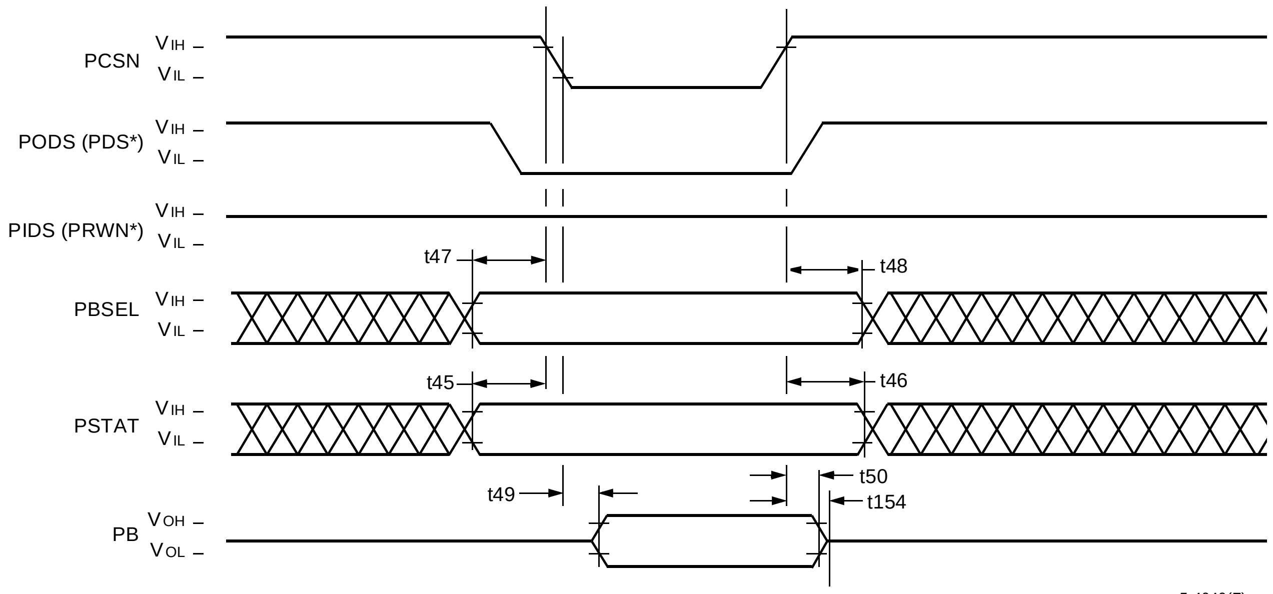
**Table 147. Timing Characteristics for PHIF16 *Motorola* Mode Signaling (Pulse Period and Flags)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t53* | PCSN/PDS to POBE† (high to high) | — | 17 | ns |
| t54* | PCSN/PDS to PIBF† (high to high) | — | 17 | 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 Figures 57 and 58. 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.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)



* *Motorola* mode signal name.

**Figure 59. PHIF16 *Intel* or *Motorola* Mode Signaling (Status Register Read) Timing Diagram**

**Table 148. Timing Requirements for PHIF16 *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 149. Timing Characteristics for PHIF16 *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.
† t154 and t50 are referenced to the rising edge of PCSN or PODS (PDS), whichever occurs first.

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

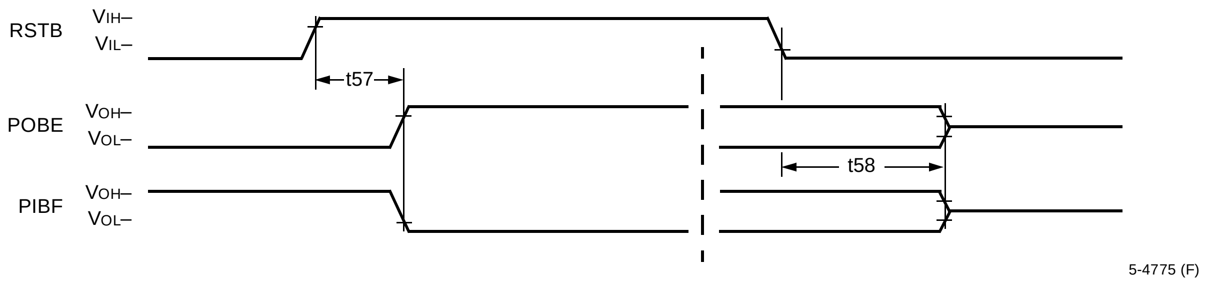
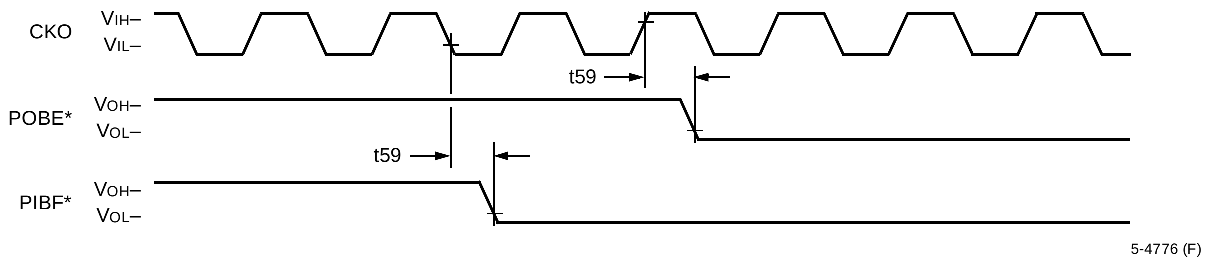


**Figure 60. PHIF16 PIBF and POBE Reset Timing Diagram**

**Table 150. PHIF16 Timing Characteristics for PIBF and POBE Reset**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t57 | RSTB Disable to POBE/PIBF* (high to valid) | — | 25 | ns |
| t58 | RSTB Enable to POBE/PIBF* (low to invalid) | 3 | 25 | 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.



* 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.

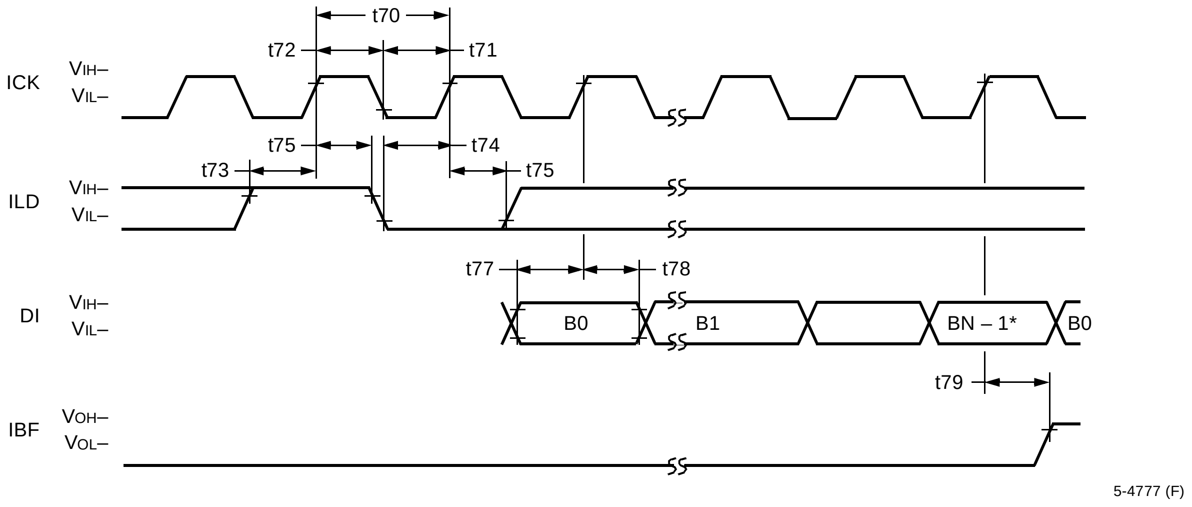
**Figure 61. PHIF16 PIBF and POBE Disable Timing Diagram**

**Table 151. PHIF16 Timing Characteristics for PIBF and POBE Disable**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t59 | CKO to POBE/PIBF Disable (high/low to disable) | — | 20 | ns |

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

### 11.9 Serial I/O Specifications



* N = 16 or 8 bits.

**Figure 62. SIO Passive Mode Input Timing Diagram**

**Table 152. Timing Requirements for Serial Inputs (Passive Mode)**

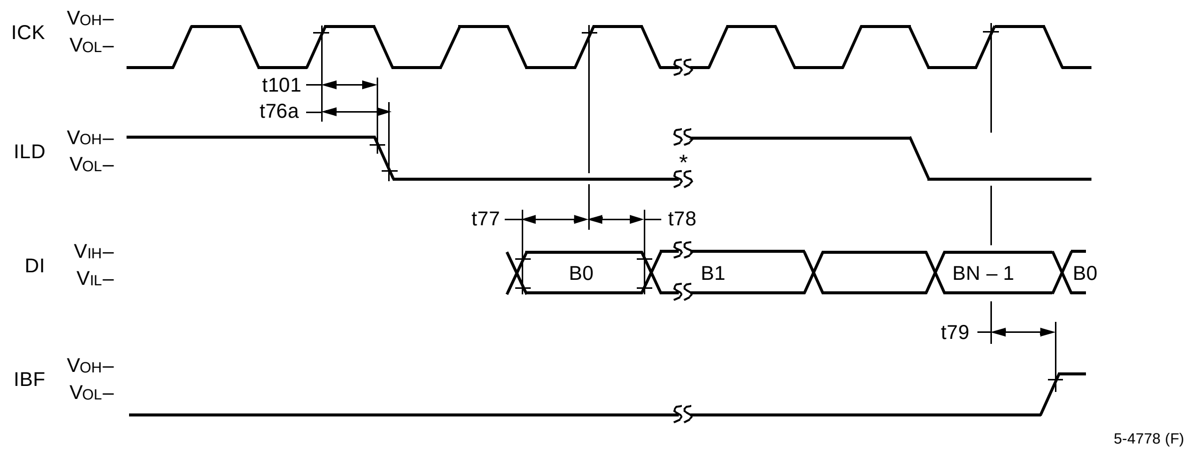
| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t70 | Clock Period (high to high)* | 40 | —† | ns |
| t71 | Clock Low Time (low to high) | 18 | — | ns |
| t72 | Clock High Time (high to low) | 18 | — | ns |
| t73 | Load High Setup (high to high) | 8 | — | ns |
| t74 | 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 |

* For multiprocessor mode, see note in Section 11.10.
† Device is fully static; t70 is tested at 200 ns.

**Table 153. Timing Characteristics for Serial Outputs (Passive Mode)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t79 | IBF Delay (high to high) | — | 35 | ns |

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)



* ILD goes high during bit 6 (of 0:15), N = 8 or 16.

**Figure 63. SIO Active Mode Input Timing Diagram**

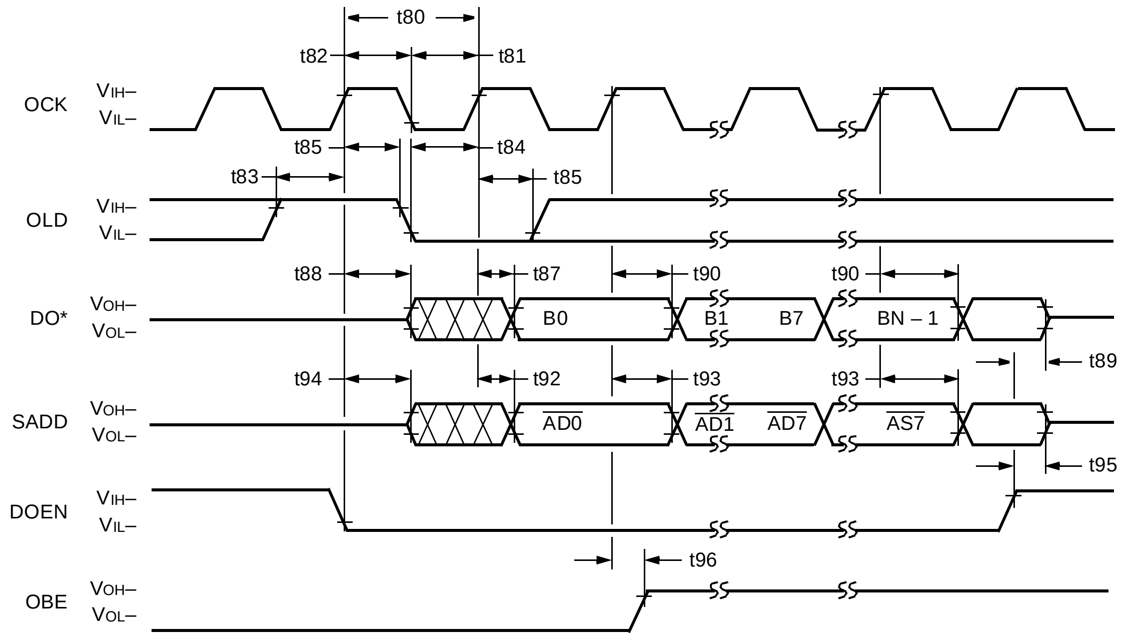
**Table 154. Timing Requirements for Serial Inputs (Active Mode)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t77 | Data Setup (valid to high) | 7 | — | ns |
| t78 | Data Hold (high to invalid) | 0 | — | ns |

**Table 155. Timing Characteristics for Serial Outputs (Active Mode)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t76a | ILD Delay (high to low) | — | 35 | ns |
| t101 | ILD Hold (high to invalid) | 1 | — | ns |
| t79 | IBF Delay (high to high) | — | 35 | ns |

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)



* See **sioc** register, MSB1 field, to determine if B0 is the MSB or LSB. See **sioc** register, ILEN1 field, to determine if the DO word length is 8 bits or 16 bits.

**Figure 64. SIO Passive Mode Output Timing Diagram**

**Table 156. Timing Requirements for Serial Inputs (Passive Mode)**

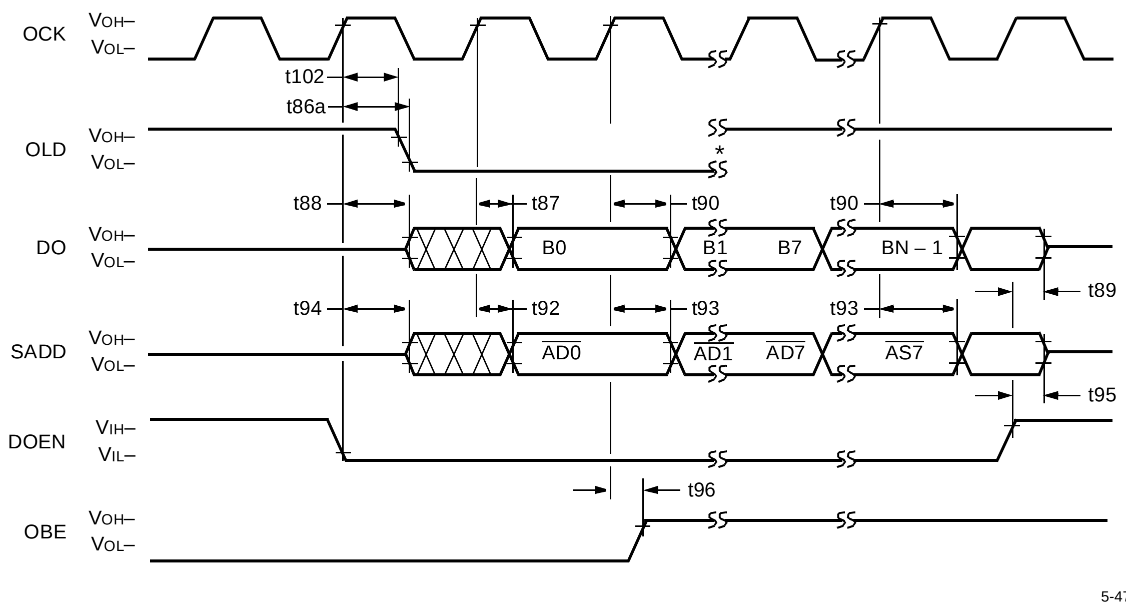
| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t80 | Clock Period (high to high)* | 40 | —† | ns |
| t81 | Clock Low Time (low to high) | 18 | — | ns |
| t82 | Clock High Time (high to low) | 18 | — | ns |
| t83 | Load High Setup (high to high) | 8 | — | ns |
| t84 | Load Low Setup (low to high) | 8 | — | ns |
| t85 | Load Hold (high to invalid) | 0 | — | ns |

* For multiprocessor mode, see note in Section 11.10.
† Device is fully static; t80 is tested at 200 ns.

**Table 157. Timing Characteristics for Serial Outputs (Passive Mode)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t87 | Data Delay (high to valid) | — | 35 | ns |
| t88 | Enable Data Delay (low to active) | — | 35 | ns |
| t89 | Disable Data Delay (high to 3-state) | — | 35 | ns |
| t90 | Data Hold (high to invalid) | 1 | — | ns |
| t92 | Address Delay (high to valid) | — | 35 | ns |
| t93 | Address Hold (high to invalid) | 1 | — | ns |
| t94 | Enable Delay (low to active) | — | 35 | ns |
| t95 | Disable Delay (high to 3-state) | — | 35 | ns |
| t96 | OBE Delay (high to high) | — | 35 | ns |

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)



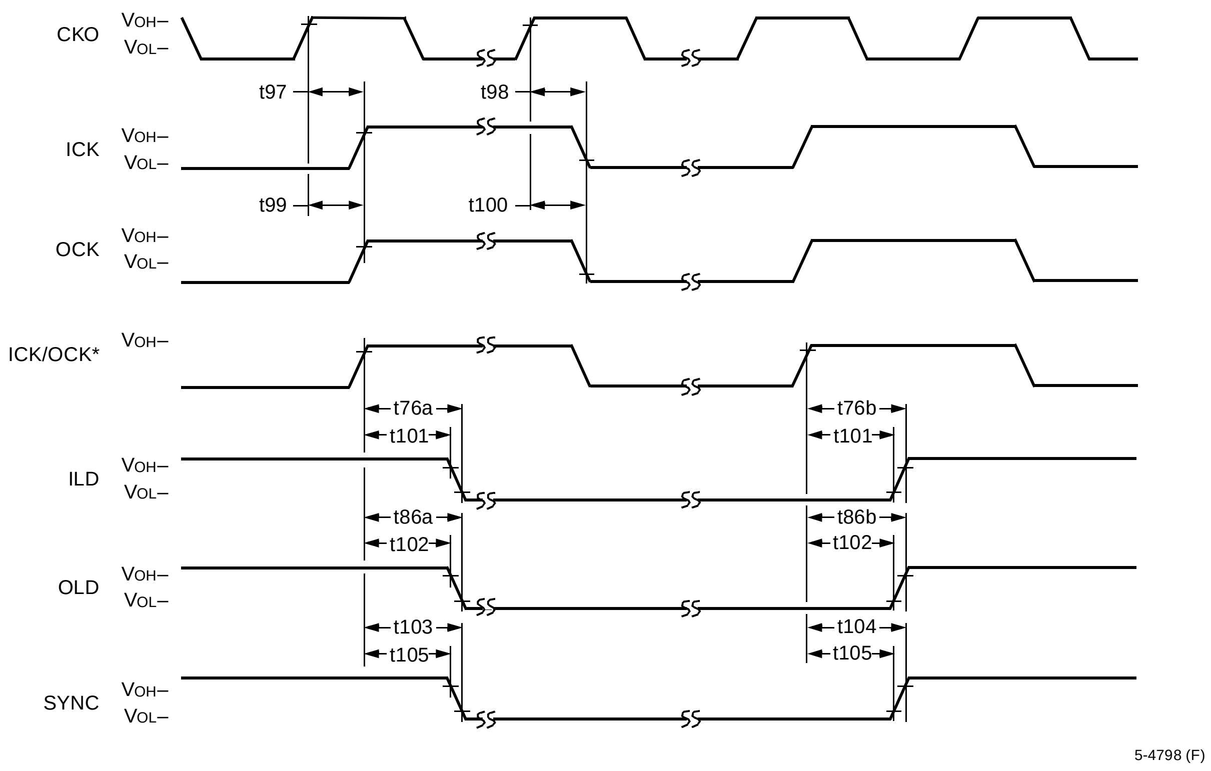
* OLD goes high at the end of bit 6 of 0:15.

**Figure 65. SIO Active Mode Output Timing Diagram**

**Table 158. Timing Characteristics for Serial Output (Active Mode)**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t86a | OLD Delay (high to low) | — | 35 | ns |
| t102 | OLD Hold (high to invalid) | 1 | — | ns |
| t87 | Data Delay (high to valid) | — | 35 | ns |
| t88 | Enable Data Delay (low to active) | — | 35 | ns |
| t89 | Disable Data Delay (high to 3-state) | — | 35 | ns |
| t90 | Data Hold (high to invalid) | 1 | — | ns |
| t92 | Address Delay (high to valid) | — | 35 | ns |
| t93 | Address Hold (high to invalid) | 1 | — | ns |
| t94 | Enable Delay (low to active) | — | 35 | ns |
| t95 | Disable Delay (high to 3-state) | — | 35 | ns |
| t96 | OBE Delay (high to high) | — | 35 | ns |

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)



* See **sioc** register, LD1 field.

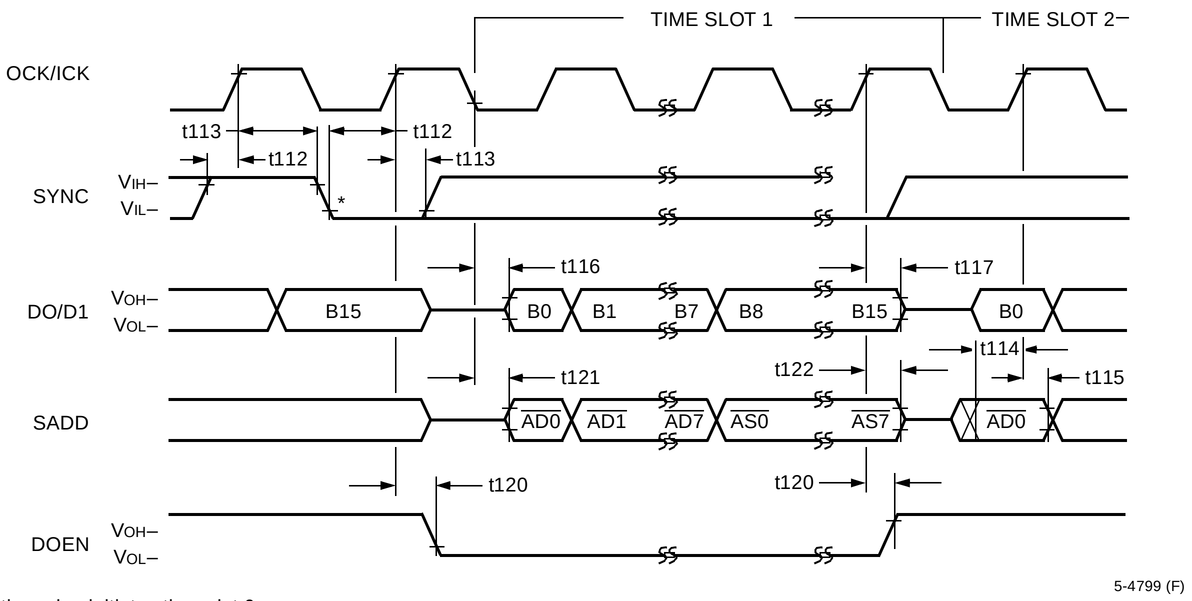
**Figure 66. SIO Active Mode Clock Timing**

**Table 159. Timing Characteristics for Signal Generation**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t97 | ICK Delay (high to high) | — | 18 | ns |
| t98 | ICK Delay (high to low) | — | 18 | ns |
| t99 | OCK Delay (high to high) | — | 18 | ns |
| t100 | OCK Delay (high to low) | — | 18 | ns |
| t76a | ILD Delay (high to low) | — | 35 | ns |
| t76b | ILD Delay (high to high) | — | 35 | ns |
| t101 | ILD Hold (high to invalid) | 1 | — | ns |
| t86a | OLD Delay (high to low) | — | 35 | ns |
| t86b | OLD Delay (high to high) | — | 35 | ns |
| t102 | OLD Hold (high to invalid) | 1 | — | ns |
| t103 | SYNC Delay (high to low) | — | 35 | ns |
| t104 | SYNC Delay (high to high) | — | 35 | ns |
| t105 | SYNC Hold (high to invalid) | 1 | — | ns |

## 11 Timing Characteristics and Requirements for 3 V Operation (continued)

### 11.10 Multiprocessor Communication



* Negative edge initiates time slot 0.

**Figure 67. SIO Multiprocessor Timing Diagram**

**Note:** All serial I/O timing requirements and characteristics still apply except the minimum clock period in passive multiprocessor mode, assuming 50% duty cycle, is calculated as (t77 + t116) x 2.

**Table 160. Timing Requirements for SIO Multiprocessor Communication**

| Abbreviated Reference | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t112 | Sync Setup (high/low to high) | 35 | — | ns |
| t113 | Sync Hold (high to high/low) | 0 | — | ns |
| t114 | Address Setup (valid to high) | 12 | — | ns |
| t115 | Address Hold (high to invalid) | 0 | — | ns |

**Table 161. Timing Characteristics for SIO Multiprocessor Communication**

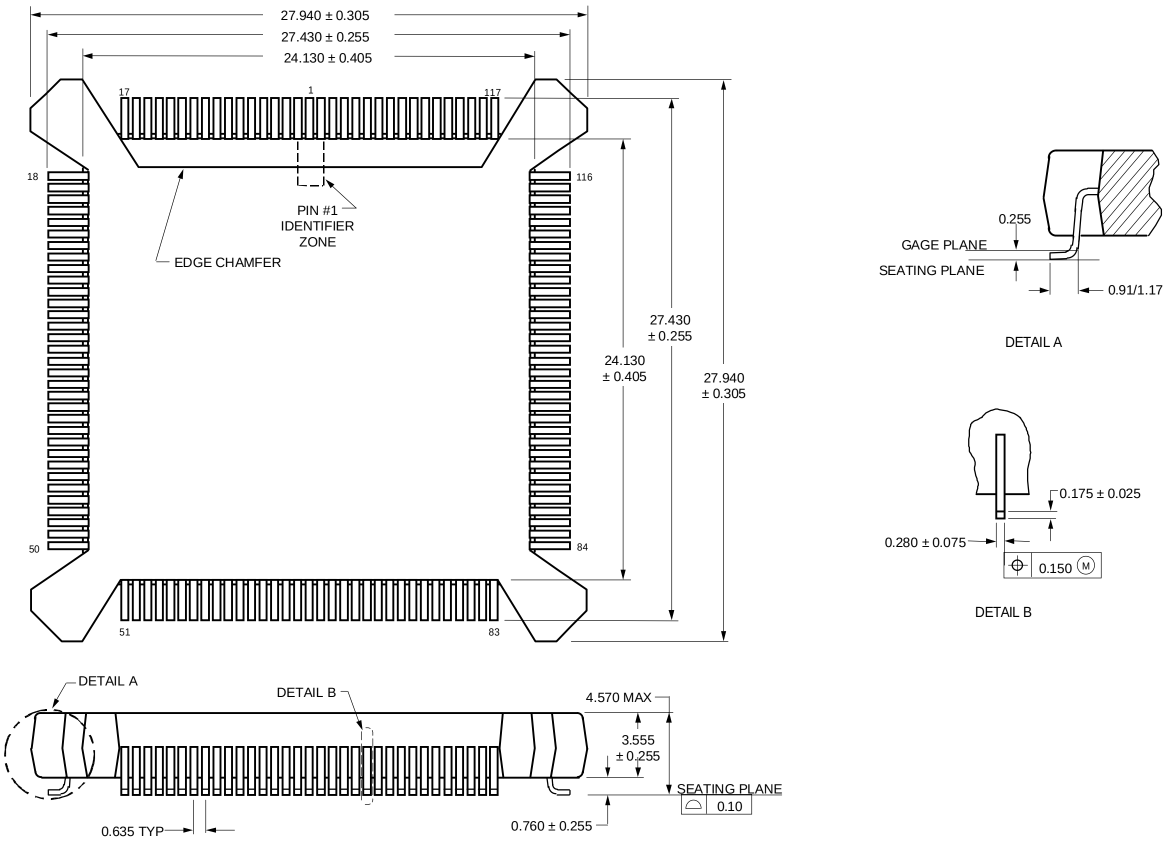
| Abbreviated Reference* | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| t116 | Data Delay (bit 0 only) (low to valid) | — | 35 | ns |
| t117 | Data Disable Delay (high to 3-state) | — | 30 | ns |
| t120 | DOEN Valid Delay (high to valid) | — | 25 | ns |
| t121 | Address Delay (bit 0 only) (low to valid) | — | 35 | ns |
| t122 | Address Disable Delay (high to 3-state) | — | 30 | ns |

* With capacitance load on ICK, OCK, DO, SYNC, and SADD = 100 pF, add 4 ns to t116—t122.

## 12 Outline Diagrams

### 12.1 132-Pin BQFP Outline Diagram

All dimensions are in millimeters.

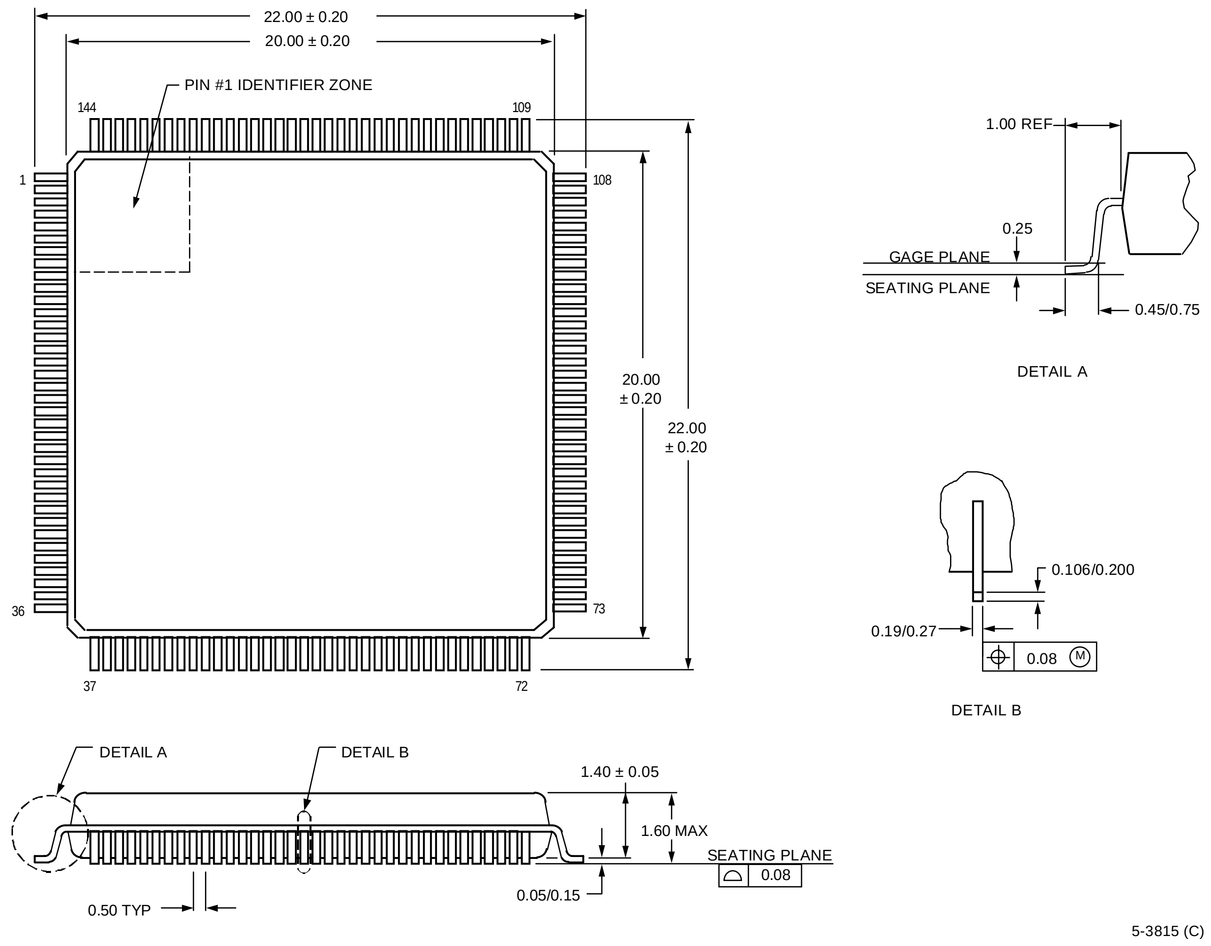


5-2586 (C)

## 12 Outline Diagrams (continued)

### 12.2 144-Pin TQFP Outline Diagram

All dimensions are in millimeters.



## Notes

## Notes

## Notes

microelectronics group

Lucent Technologies
Bell Labs Innovations

June 1998
DS97-321WDSP (Replaces DS95-224WDSP)