T320C2700B0 Customizable DSP (cDSP™) System Configuration Design Guide

Literature Number: SPRU272A
December 1998

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Preface

Read This First

About This Manual

This book provides system configurations and signal connections to enable you to design with the T320C2700B0 Customizable DSP (cDSP™) core. It also describes considerations you may want to take into account in designing your logic.

This book assumes that you are familiar with the ’C2700B0 architecture and memory interface. For details about the ’C2700B0 architecture, see the *TMS320C27xx DSP CPU and Instruction Set Reference Guide*. For details about the ’C2700B0 memory interface, see the *T320C2700 Customizable Digital Signal Processor (cDSP) Core* data sheet and the *T320C2700B0 Customizable Digital Signal Processor (cDSP) Core (TSC6000 ASIC Libraries)* data sheet.

How to Use This Manual

The following table summarizes this book’s information by topic, showing you which chapters contain the information you are interested in.

If you are looking for information about:	**Turn to:**
Overview of the T320C2700B0	Chapter 1, *Introduction*
Memory configuration	Chapter 2, *Memory Configurations*
External interface	Chapter 3, *External Interface (XINTF)*
Timer	Chapter 4, *Timer*
Test requirements	Chapter 5, *T320C2700B0 Test Requirements and Considerations*
Design example	Chapter 6, *T320C2700B0 cDSP Design Example*
Electrical considerations	Chapter 7, *Electrical Considerations*
Signal descriptions	Appendix A, *Signal Descriptions (TSC6000 ASIC Library)*

If you are looking for information about:	Turn to:
Instruction set summary	Appendix B, *T320C2700B0 Instruction Set Summary*
DSPnetGEN tutorial	Appendix C, *DSPnetGEN Tutorial*

Notational Conventions

Program listings, program examples, and interactive displays are shown in a `special typeface`.

Information About Cautions

This book contains cautions.

> **This is an example of a caution statement.**
>
> **A caution statement describes a situation that could potentially damage your software or equipment.**

The information in a caution is provided for your protection. Please read each caution carefully.

Related Documentation From Texas Instruments

The following books describe the T320C2700B0 and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the book by its title and literature number.

Submicron ASIC Products Design Software Manual (literature number SRGA003) provides a design flow overview of the tools and processes of the 0.8-μm gate array series and describes in depth each particular software tool.

Submicron ASIC Products TI Design Support Software Release 4.2 for Mentor 8.5 (literature number SRUA018) describes the release notes for HP 9000/HP 700 series workstations with HP-UX 9.0x and 10.x, Sun SPARCstation 10/20 platforms, with SunOS 4.1.3 and Solaris 2.4 and 2.5.a, and Sun ULTRAsparc 1 platform with Solaris 2.5.1.

Submicron ASIC Products ASIC Compiler Environment (ACE) User's Guide (literature number SRGU003) describes the features and capabilities of the ASIC Compiler Environment (ACE) and provides view descriptions of the compiler functions available in ACE. It also contains installation information for network administrators and execution information for ACE users.

Submicron ASIC Products Design for Testability Reference Guide (literature number SRUU002) offers guidelines for developing a coherent approach to integrating testability in the design flow.

TMS320C27xx DSP CPU and Instruction Set Reference Guide (literature number SPRU220) describes the central processing unit (CPU) and the assembly language instructions of the TMS320C27xx 16-bit fixed-point digital signal processors (DSPs). It also describes emulation features available on these DSPs.

T320C2700 Customizable Digital Signal Processor (cDSP™) Core (literature number SPRS057) data sheet contains the electrical and timing specifications for these devices.

TMS320C2700–E1 Digital Signal Processor In-Circuit Emulation Device (literature number SPRS062) data sheet contains the pinout, signal descriptions, as well as electrical and timing specifications for the device.

T320C2700B0 Customizable Digital Signal Processor (cDSP™) Core (literature number SPRS069) data sheet contains the electrical and timing specifications for these devices.

TMX320C2700-E3 Digital Signal Processor In-Circuit Emulation Device (literature number SPRS068) data sheet contains the pinout, block diagram, component descriptions, timing information, electrical specifications, and mechanical package for the device.

TMS320C27xx Code Generation Tools Getting Started Guide (literature number SPRU213) describes how to install the TMS320C27xx assembly language tools and the C compiler for the TMS320C27xx device. The installation for MS-DOS™, SunOS™, and HP-UX™ 9.0x systems are covered.

TMS320C27xx Assembly Language Tools User's Guide (literature number SPRU211) describes the assembly language tools (assembler and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the TMS320C27xx device.

TMS320C27xx Optimizing C Compiler User's Guide (literature number SPRU212) describes the TMS320C27xx C compiler. This C compiler accepts ANSI standard C source code and produces TMS320 assembly language source code for the TMS320C27xx device.

TMS320C27xx Simulator Getting Started (literature number SPRU216) describes how to install the simulator and the C source debugger for the TMS320C27xx device. The installation for MS-DOS™, SunOS™, and HP-UX™ systems are covered.

TMS320C27xx Emulator Getting Started (literature number SPRU215) describes how to install the emulator software and the C source debugger for the TMS320C27xx device. The installation for MS-DOS™, SunOS™, and HP-UX™ systems are covered.

TMS320C27xx C Source Debugger User's Guide (literature number SPRU214) tells you how to invoke the TMS320C27xx emulator and simulator versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints. It also includes a tutorial that introduces basic debugger functionality.

TMS320C27xx Translation Assistant User's Guide (literature number SPRU278) describes the TMS320C27xx translation utility and how it fits in with the rest of the TMS320C27xx code development tools. It tells you how to use the translation assistant to translate code you already have for TMS320C2xx devices into code that will run on TMS320C27xx devices.

33F12.2 Embedded Flash EEPROM Product Specification

Trademarks

HP-UX is a trademark of Hewlett-Packard Company.

MS-DOS is a registered trademark of Microsoft Corporation.

Solaris and SunOS are trademarks of Sun Microsystems, Inc.

Synopsys is a trademark of Synopsys, Inc.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

320 Hotline On-line, ACE, cDSP, TI, and XDS510 are trademarks of Texas Instruments Incorporated.

If You Need Assistance . . .

❑ World-Wide Web Sites

TI Online	http://www.ti.com
Semiconductor Product Information Center (PIC)	http://www.ti.com/sc/docs/pic/home.htm
DSP Solutions	http://www.ti.com/dsps
320 Hotline On-line™	http://www.ti.com/sc/docs/dsps/support.htm

❑ North America, South America, Central America

Product Information Center (PIC)	(972) 644-5580		
TI Literature Response Center U.S.A.	(800) 477-8924		
Software Registration/Upgrades	(214) 638-0333	Fax: (214) 638-7742	
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285		
U.S. Technical Training Organization	(972) 644-5580		
DSP Hotline	(281) 274-2320	Fax: (281) 274-2324	Email: dsph@ti.com
DSP Modem BBS	(281) 274-2323		

DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:

Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32

Email: epic@ti.com

Deutsch	+49 8161 80 33 11 or +33 1 30 70 11 68	
English	+33 1 30 70 11 65	
Francais	+33 1 30 70 11 64	
Italiano	+33 1 30 70 11 67	
EPIC Modem BBS	+33 1 30 70 11 99	
European Factory Repair	+33 4 93 22 25 40	
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828
Korea DSP Modem BBS	+82 2 551 2914	
Singapore DSP Hotline		Fax: +65 390 7179
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718
Taiwan DSP Modem BBS	+886 2 376 2592	

Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/

❑ Japan

Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259
DSP Hotline	+03-3769-8735 or (INTL) 813-3769-8735	Fax: +03-3457-7071 or (INTL) 813-3457-7071
DSP BBS via Nifty-Serve	Type “Go TIASP”	

❑ Documentation

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Mail: Texas Instruments Incorporated
Technical Documentation Services, MS 702
P.O. Box 1443
Houston, Texas 77251-1443

Email: dsph@ti.com

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

Contents

Figures

Tables

Examples

Chapter 1

Introduction

This chapter introduces the digital signal processor (DSP), the application-specific integrated circuit (ASIC), the customizable digital signal processor (cDSP™), the cDSP design flow, and the ’C2700B0 cDSP device. See Appendix A, *Signal Descriptions (TSC4000 ASIC Library)*, for a description of the ’C2700B0 signals.

Topic **Page**

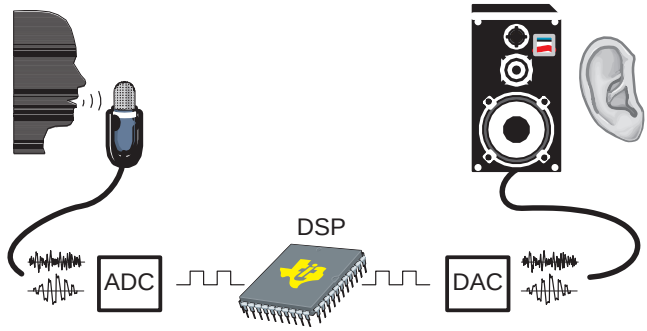
1.1 What Is a Digital Signal Processor (DSP)?

A digital signal processor (DSP) is a single-chip microcomputer that manipulates digital signals prescribed by the algorithm you apply. Many of these signals begin as analog or continuously variable information. Examples of this information are sound, temperature, light, pressure, position, and speed. Electronic sensors convert the analog information into electrical signals that are then passed through an analog-to-digital (ADC) converter to digitize them for use by the DSP.

When a signal has been manipulated by a DSP, it remains digital. If needed, the digital signal can be converted back into an analog signal by a digital-to-analog (DAC) converter.

A simple digital signal processing system is shown in Figure 1–1.

Figure 1–1. A Digital Signal Processing System



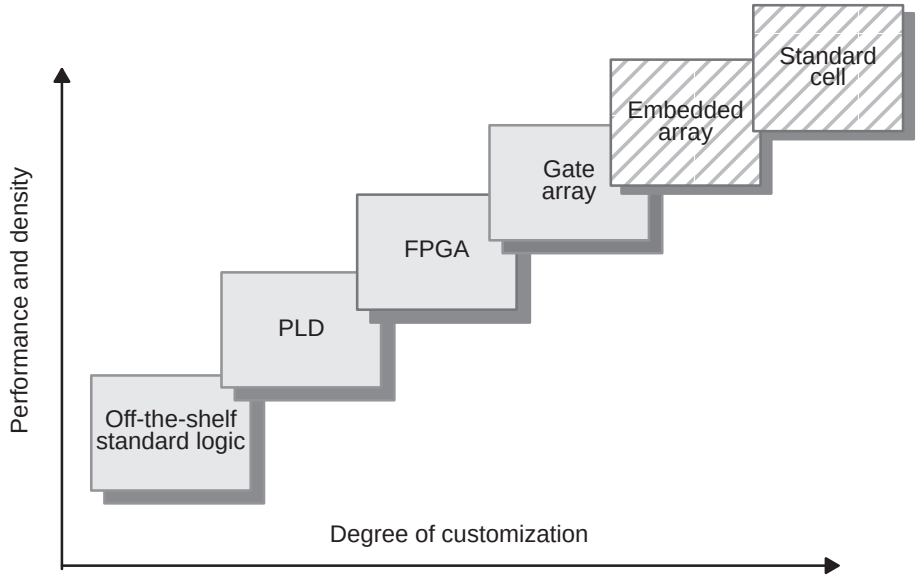
1.2 What Is an Application-Specific Integrated Circuit (ASIC)?

Unlike DSPs, which are standard parts, application-specific integrated circuits (ASICs) use the same building blocks to create a wide variety of functions and can be customized to implement user-defined logic.

ASIC technologies include devices like programmable logic devices (PLDs), field programmable gate arrays (FPGAs), gate arrays, embedded arrays, and standard cells. The main feature that distinguishes one ASIC technology from another is the freedom you have to change the structure of the device or the available customizability.

Figure 1–2 shows the ASIC devices from the lowest level of performance and customization to the highest.

Figure 1–2. Performance and Density of ASIC Devices Versus Degree of Customization



1.3 What is a Customizable Digital Signal Processor (cDSP)?

A cDSP is an ASIC device that has a DSP core. The cDSP allows you to achieve a high degree of system integration, low power consumption, and high performance.

Combined with TI's development tools and support, a cDSP provides a full-service solution for achieving increased levels of integration for DSP systems. This customization capability opens the door for developing high-volume end equipment. TI's extensive development tools, third-party tools, and software libraries for industry-standard algorithms are available for cDSPs, just as they are for the TMS320 DSPs.

The advantages of cDSPs over DSPs are:

- Each cDSP is optimized for the particular application.
  - Minimizes engineering effort
  - Results in fast time to market
- The combination of technologies gives improved levels of integration, which:
  - Enables major performance improvements
  - Reduces overall system cost
  - Meets system size restrictions
  - Reduces system power consumption
  - Reduces system noise
  - Increases reliability

1.4 DSP/ASIC/cDSP Development Flows

Because cDSPs are the product of integrating a DSP within an ASIC device, you must use the development tools and design flows of both technologies to successfully implement a cDSP design. Table 1–1 compares simplified versions of the DSP/ASIC/cDSP development flows.

Table 1–1. Simplified Version of DSP/ASIC/cDSP Development Flows

Simplified Flow (Steps)	TI DSP	TI ASIC	TI cDSP
cDSP requirements: System definition System partition Architecture design	√	√	√
Software development: Algorithm description Flow diagram High-level language (HLL) description HLL description simulator	√		√
Code preparation: Assembly language Assembler Link to target	√		√
Design entry: Hardware description and logic synthesis Schematic capture and gate-level netlist Design database creation		√	√
Prelayout verification: Input stimuli creation Prelayout simulations Test vector generation		√	√
Physical layout (from design database): Place and route Interconnect and delay extraction Update design database with physical layout data		√	√
Postlayout verification: Back annotation Postlayout simulations Test vector verification		√	√
Prototype fabrication and testing: Pattern generation Photomask tooling Wafer processing Die/package assembly Testing		√	√

1.5 Functions of DSP and ASIC Designers

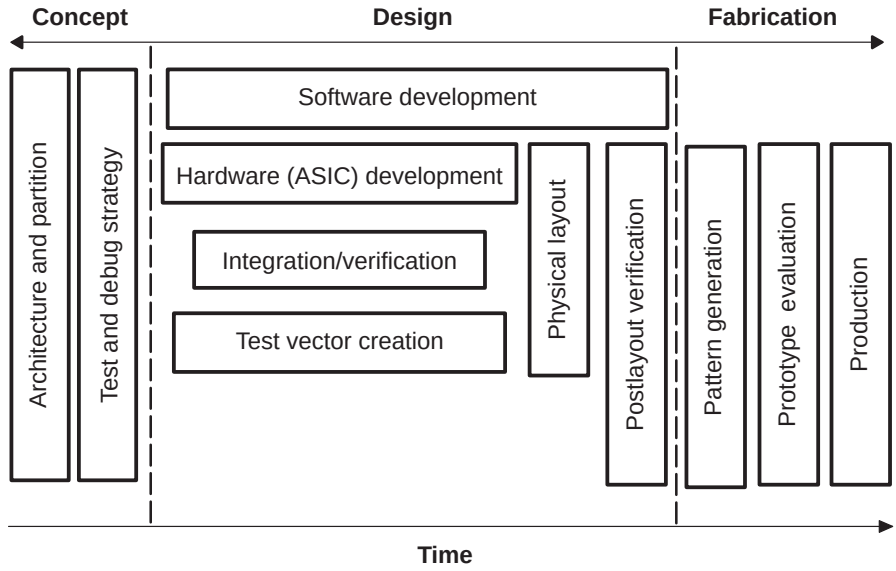
When faced with the task of designing a system, DSP designers typically implement software solutions that involve writing code, using models, and interfacing to other parts of the system. ASIC designers, on the other hand, typically implement hardware solutions that involve gates, test vectors, layout, test equipment, etc. A cDSP designer needs to know both perspectives. The following chart shows the two perspectives:

DSP	ASIC
Partition system and design architecture	Partition system and design architecture
Use simulator to verify functionality of the system design	Create and verify logic using CAD tools like schematic capture, logic synthesis, and simulation
Verify cDSP operation in the system	Generate test vectors using TI's Design Support Software (TIDSS)
	Verify prototypes and approve production orders

1.6 Overview of the cDSP Design Process

The cDSP design process is similar to the ASIC design process, except that the cDSP involves software *and* hardware and the ASIC flow involves primarily hardware. The cDSP process can be thought of in the typical three phases of a new product's process: concept, design, and fabrication. Figure 1–3 shows the steps in each phase of the process.

Figure 1–3. cDSP Design Phases



1.7 cDSP Design Teams

When a cDSP application is identified, TI forms a team that works on the cDSP project through all three cDSP design phases. These TI personnel serve as permanent members of the design team:

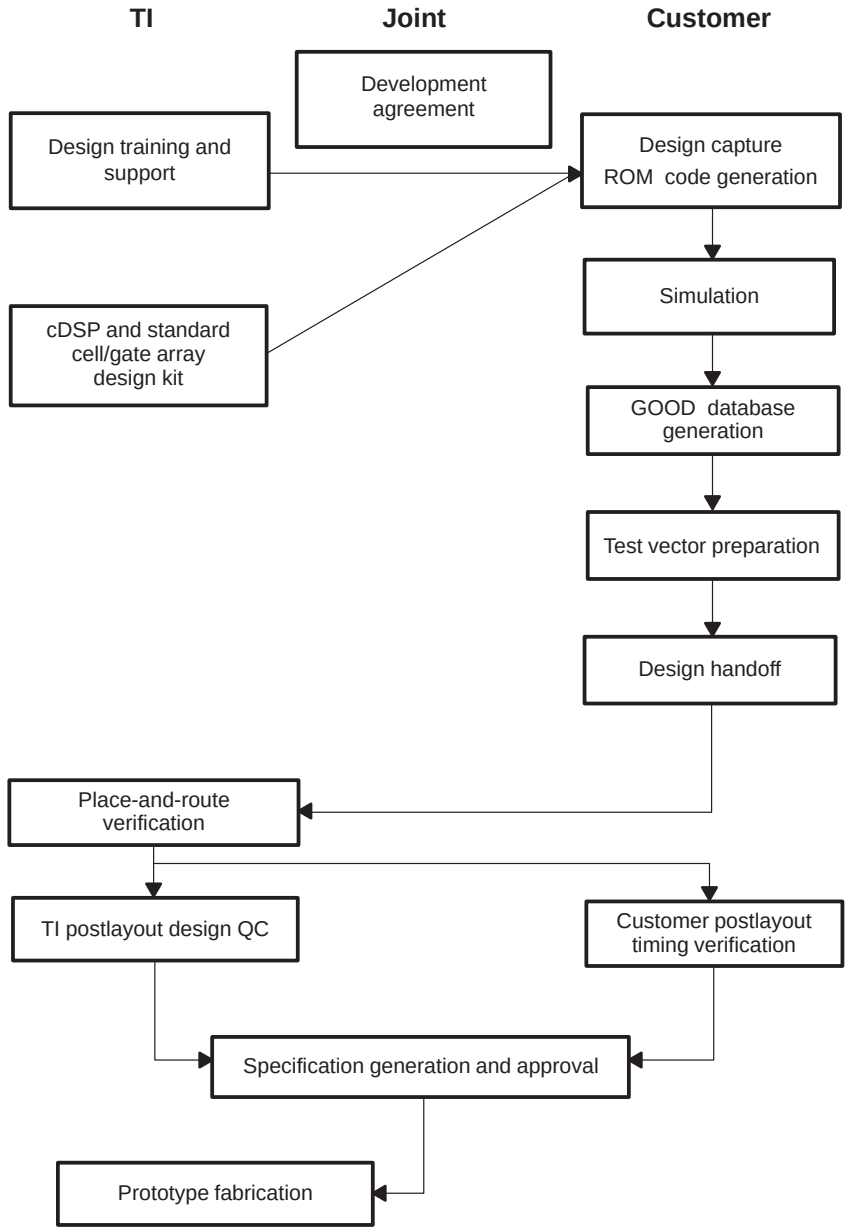
- Technical sales representative (TSR)
- Technical staff (TS)
- cDSP program manager

During the appropriate phases of the cDSP design, the following groups are represented on the design team as well:

- DSP marketing
- DSP applications
- DSP product engineering
- DSP planning
- ASIC physical design
- ASIC applications
- ASIC EE applications

A joint development agreement between the customer and TI ensures the timely execution of all steps in the design flow. It also defines the responsibilities for identifying tasks to successfully complete the cDSP design. Experience shows that the allocation of team responsibilities shown in Figure 1–4 works well for most design projects.

Figure 1–4. cDSP Design Team Responsibilities



1.8 Overview of the T320C2700B0

The ’C2700B0 cDSP core is a low cost, 16-bit fixed-point DSP optimized for mass-storage mechanical and interface control applications. This device draws from the best features of digital signal processing, reduced instruction set computing (RISC), and microcontroller architectures, firmware, and tool sets. The DSP features include Harvard architecture and circular addressing. The RISC features are single-cycle instruction execution, register-to-register operations, and modified Harvard architecture (can be used in Von Neumann mode). The microcontroller is easy to use because it features an intuitive instruction set, byte packing and unpacking, and good bit manipulation.

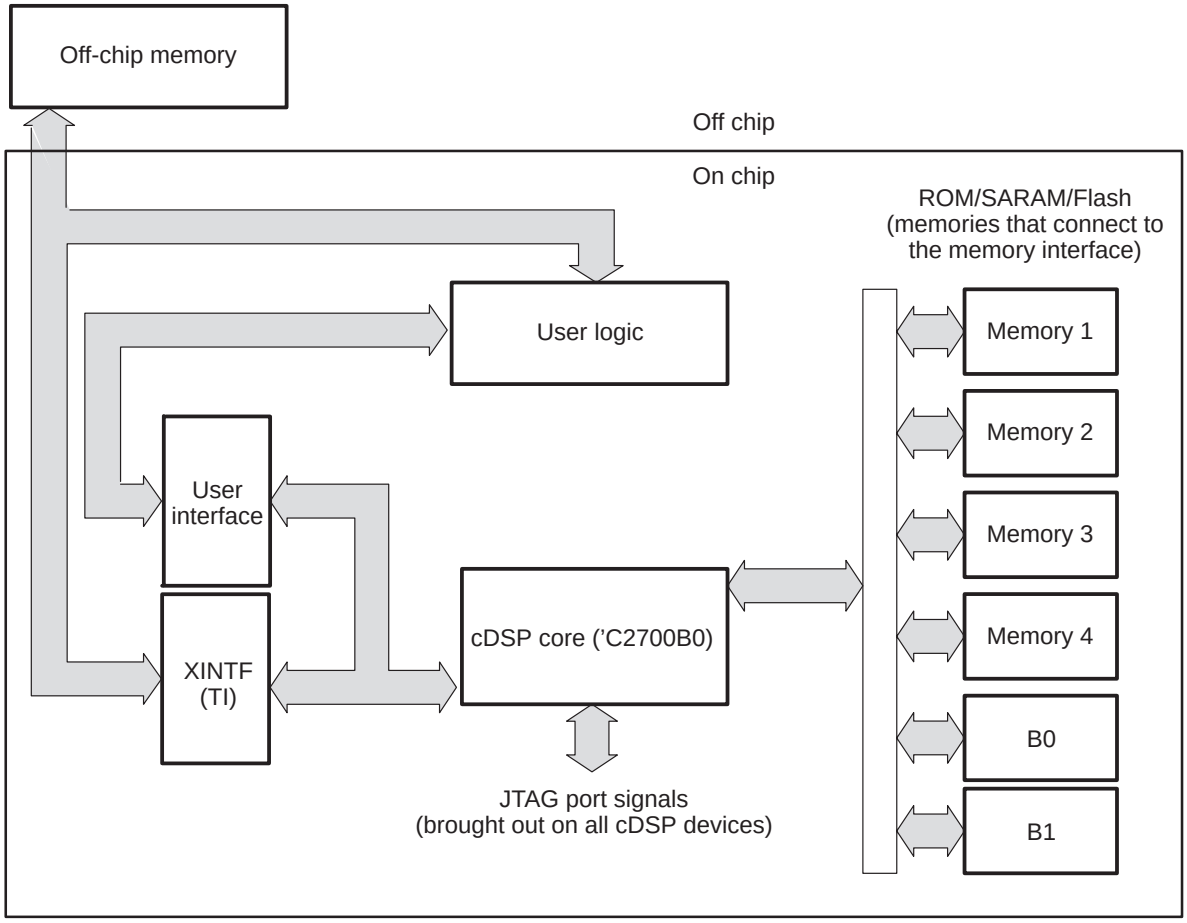
The core consists of a central processing unit (CPU), emulation logic, and signals for interfacing with memory and peripherals. It also includes six interface buses, three 22-bit address buses, and three 32-bit data buses. The ’C2700B0 uses a modified Harvard architecture. This uses multiple memory spaces to enable instruction and data fetches to be performed in parallel, allowing single-cycle instructions. The CPU can read instructions and data while it writes data simultaneously to maintain the single-cycle instruction operation across the pipeline. The CPU does this over the six separate address/data buses. The 22-bit address buses, program (PAB), data read (DRAB), and data write (DWAB), allow a total of 4M $\times$ 16 memory in both program and data space. The core does not contain memory, a clock generator, or peripheral devices.

The ’C2700B0 cDSP core is designed to be embedded into a cDSP device as a hardware macro. The TI ASIC design environment enables logic designers to combine custom logic with the ’C2700B0 cDSP core and other hardware macros to create a cDSP.

Figure 1–5 shows an example of a cDSP device using the ’C2700B0. A typical cDSP device consists of a ’C2700B0 cDSP core, B0 and B1 memory blocks, memory modules, interface bridges, and custom logic. See the *TMS320C27xx DSP CPU and Instruction Set Reference Guide* for details of the ’C2700B0 architecture.

The ’C2700B0 cDSP environment uses three types of memory modules: single-access random-access memory (SARAM), read-only memory (ROM), and Flash memory. The memory modules typically consist of a memory wrapper and a memory core. A memory wrapper is a unit that performs arbitration of CPU requests and acts as an interface between the ’C2700B0 memory interface signals and the memory core (composed of ROM, SARAM, or Flash memories). A memory interface consists of a set of signals through which the core communicates with memories and other external components in the memory space, such as the memory wrappers and peripherals.

Figure 1–5. Typical cDSP Configuration



User-defined interfaces and the TI-provided external interface (XINTF) map memory interface signals to the cDSP core and act as adapters for external memory interface components (components connected to the memory interface) and user logic. A memory interface component is any component external to the 'C2700B0 core such as a memory core or peripheral. Interface bridges are usually more complex in functionality than memory wrappers.

Basic debug access to the DSP core on a 'C2700B0 cDSP device is through the IEEE 1149.1 JTAG port. The JTAG port consists of the standard JTAG signals (TRST, TMS, TDI, TDO, and TCK) and TI's extensions (EMU0 and EMU1).

Chapter 2

Memory Configurations

This chapter describes the ’C2700B0 memory configurations, including clocked ROM (CROM) and single-access RAM (SARAM). It also shows a memory configuration example for a CROM and dual SARAM memory map.

Topic **Page**

2.1 Memory Overview

Memories can either be mapped in data space or in both program and data space. If you are using the ’C2700B0 C compiler, you cannot have a memory that is located only in program space because the compiler loads data tables into data space. Any memory block mapped in program space must have an equivalent data space mirror. A mirror occurs when two different memory addresses access the same physical memory. Because these two spaces map to the same physical memory, you must be careful not to overwrite needed program memory by writing to mirrored data memory. The mirrored data space does not necessarily need to start at the same memory address as the program space. The data-space start address (DSTRT) and program-space start address (PSTRT) buses can have different values (explained further in section 2.2.1, *CROM Configuration Guidelines,* on page 2-4).

A conflict occurs if there are two or more simultaneous requests to the same memory module. In this case, the memory wrapper uses the output ready signals to inform the ’C2700B0 core that it cannot service any new request. Whenever there is a request from the ’C2700B0 core, the memory wrapper decodes the address bus to decide whether the memory module is selected or not. If there is more than one request to the memory module, the memory wrapper stores the requests and services them one by one, according to the following priority:

1) Any write
2) Data read
3) Program read

Any stored request has higher priority than a new request. See Table 2–1 for bus priority examples.

The ’C2700B0 memory space is 16-bit word addressable, but has the option to request a 32-bit read or write. A memory module’s starting address must be a multiple of its size (binary boundary) to be properly decoded. For example, a 16K × 16 memory block must begin at address 0000h, 4000h, 8000h, C000h, ..., 3F C000h.

Note:

Address 0000h is reserved for B0 and B1 SARAM. See section 2.4 on page 2-21 for more details.

Addresses 800h–C00h are reserved for emulation registers and peripheral registers. See section 2.4 for more information.

Any unused input pins to the ’C2700B0 core or a memory wrapper must be driven by a logic 1 or a logic 0 signal to minimize noise and power dissipation.

For information on connecting your logic to the memory interface, see section 5.3.3 on page 5-12.

For a description of the memory interface signals, see section A.1.1 on page A-2.

Table 2–1. Bus Priorities

PRDB[31:0]	DRDB[31:0]	DWDB[31:0]	Priority
	Data read	Any write	Any write Data read
Program read	Data read		Data read Program read
Program read		Any write	Any write Program read
	Data read	Any write	Any write Data read
Program read	Data read	Any write	Any write Data read Program read

2.2 Clocked ROM (CROM)

A generic ROM wrapper interfaces the ’C2700B0 core and the TI ASIC compiler environment (ACE™) clocked ROM (CROM) cores. This wrapper supports 256, 512, 1K, 2K, 4K, 8K, 16K, and 32K 32-bit-wide longwords of single 32-bit ROM core mapped anywhere in the 4M × 16 program space. Mirrors occur for ROM cores less than 256 × 32. For more information on mirrors, see section 2.1, *Memory Overview,* on page 2-2.

The following are the features and options of the CROM wrapper:

- Designed in VHSIC hardware description language–register transfer level (VHDL–RTL) to allow synthesis into any number of TI ASIC technologies (for example, TSC6000)
- A gate-level, timing-optimized netlist out of Synopsys™
- Designed to interface from the ’C2700B0 memory interface to the MX ROM cores

For a description of the CROM wrapper signals, see section A.2 on page A-22.

2.2.1 CROM Configuration Guidelines

The PON pin on the CROM wrapper should be connected to logic 1 or logic 0 to determine what memory-space pins to activate on the wrapper. An active (high) PON indicates the CROM is in program space. If you are using the ’C2700B0 C compiler, you cannot have a memory that is located only in program space.

The 13-bit PSTRT[12:0] and DSTRT[12:0] buses on the CROM wrapper are compared to PAB[21:9] and DRAB[21:9], respectively, from the ’C2700B0 core to determine if the current read access is to the wrapper. PSTRT and DSTRT are configured according to the size of the CROM core. If the CROM core is greater than 256 × 32, the lower bits of PSTRT and DSTRT must be connected to the lower bits of PAB[21:9] and DRAB[21:9], respectively.

The address bus, MA[15:1], connects the CROM wrapper to the CROM core. The lowest bit (MA0) is unconnected because the CROM core is only 32-bit accessible. The wrapper logic determines the alignment and size of a program or data read based on the PRDS0, PRDS1, DRDS0, and DRDS1 signals (see section A.1.1, *Memory Interface Signals* on page A-2). If the CROM core is smaller than 32K × 32, the upper bits of MA[15:1] are left unconnected.

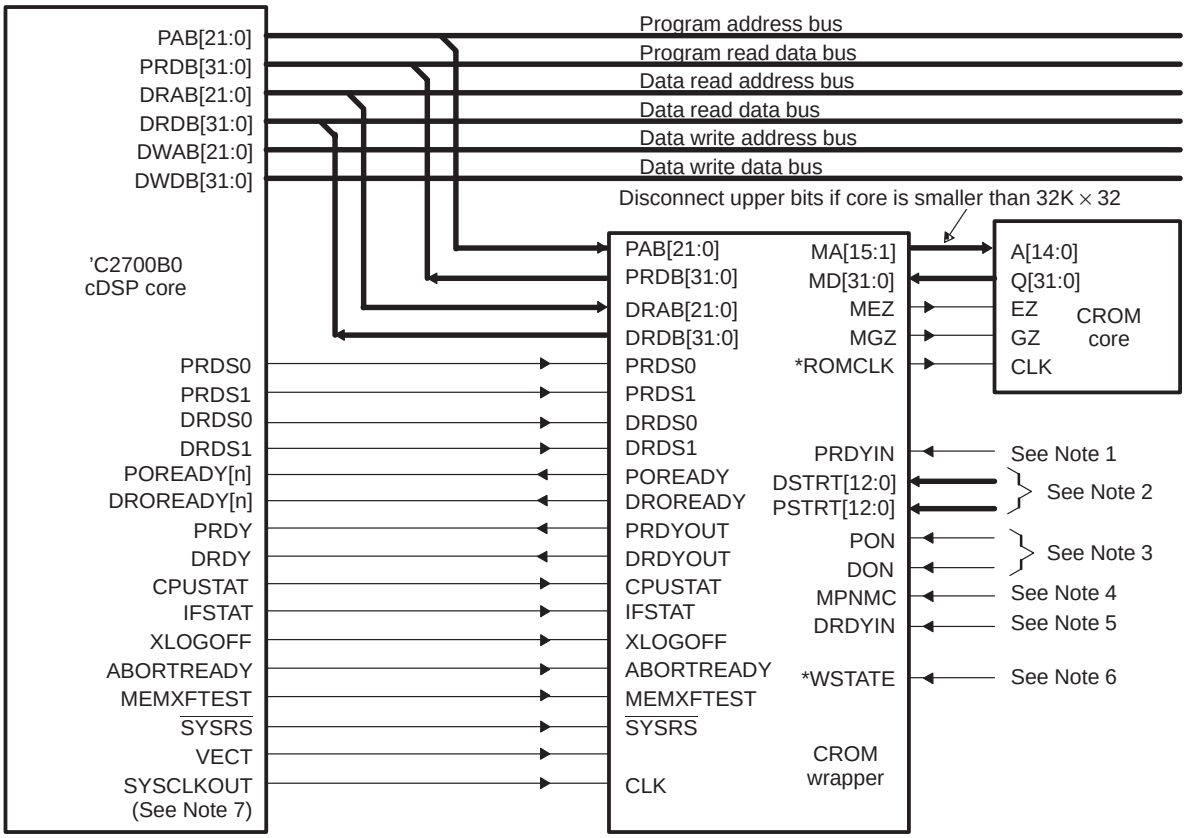
If a CROM core is 128 × 32, PAB[8] is unaccounted for on PSTRT[12:0] and MA[7:1], and a mirror occurs at 100h from the block’s starting address. Table 2–2 summarizes these rules for PSTRT and MA. These PSTRT concepts apply to data space configurations as well.

Table 2–2. Example CROM Configurations

CROM Core Size	Starting Address	PSTRT Configuration	Address Bus to CROM Core	Mirrors
128 × 32	01 0000h	PSTRT[12:0] → 00 0001 0000 000	MA[7:1] → A[6:0]	01 0100h
256 × 32	01 0000h	PSTRT[12:0] → 00 0001 0000 000	MA[8:1] → A[7:0]	–
512 × 32	01 0000h	PSTRT[12:1] → 00 0001 0000 00 PSTRT[0] → PAB[9]	MA[9:1] → A[8:0]	–
1K × 32	01 0000h	PSTRT[11:2] → 00 0001 0000 0 PSTRT[1:0] → PAB[10:9]	MA[10:1] → A[9:0]	–
2K × 32	01 0000h	PSTRT[12:3] → 00 0001 0000 PSTRT[2:0] → PAB[11:9]	MA[11:1] → A[10:0]	–
4K × 32	01 0000h	PSTRT[12:4] → 00 0001 000 PSTRT[3:0] → PAB[12:9]	MA[12:1] → A[11:0]	–
8K × 32	01 0000h	PSTRT[12:5] → 00 0001 00 PSTRT[4:0] → PAB[13:9]	MA(13:1) → A[12:0]	–
16K × 32	01 0000h	PSTRT[12:6] → 00 0001 0 PSTRT[5:0] → PAB[14:9]	MA[14:1] → A[13:0]	–
32K × 32	01 0000h	PSTRT[12:7] → 00 0001 PSTRT[6:0] → PAB([5:9]	MA[15:1] → A[14:0]	–

See Figure 2–1 for a detailed CROM configuration block diagram. Section 2.5, *Example Memory Configuration*, on page 2-23 contains an example 'C2700B0 core/memory setup.

Figure 2–1. CROM Connections to the T320C2700B0 Core Block Diagram



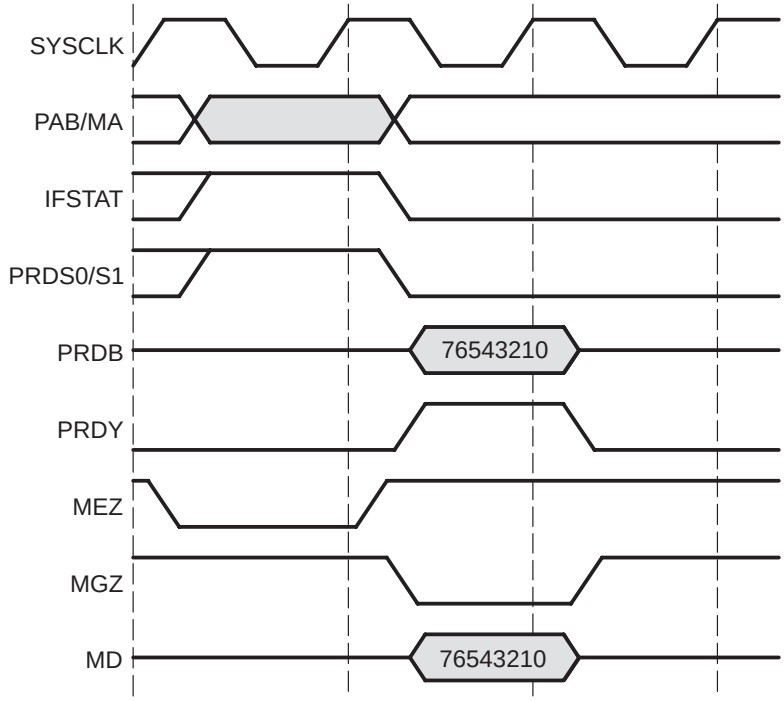
Notes:

1) Daisy-chained to PRDYOUT of another wrapper for multiple program memory blocks, tied low if CROM wrapper first block
2) See section 2.2.1, *CROM Configuration Guidelines*, for PSTRT and DSTRT connection rules.
3) See section 2.2.1, *CROM Configuration Guidelines*, for PON and DON connection rules.
4) From ASIC logic (if low, ROM is enabled; if high, ROM is disabled); ignore when MEMXFTEST is high
5) Daisy-chained to DRDYOUT of another wrapper for multiple data memory blocks; tied low if CROM wrapper is first block.
6) When tied low, wrapper assumes 0 wait-state accesses; when tied high, wrapper assumes 1 wait-state access.
7) SYSCLKOUT is not connected directly to CLK on the CROM wrapper, SYSCLKOUT must go through a clock tree synthesis (CTS) buffer.
8) Signals denoted * exist only in the dual memory space version of the CROM wrapper.
9) See Figure 5–2 on page 5-8 for connection of scan chain signals: SCEN, SCIN, and SCOUT.

2.2.2 CROM Timing Diagrams

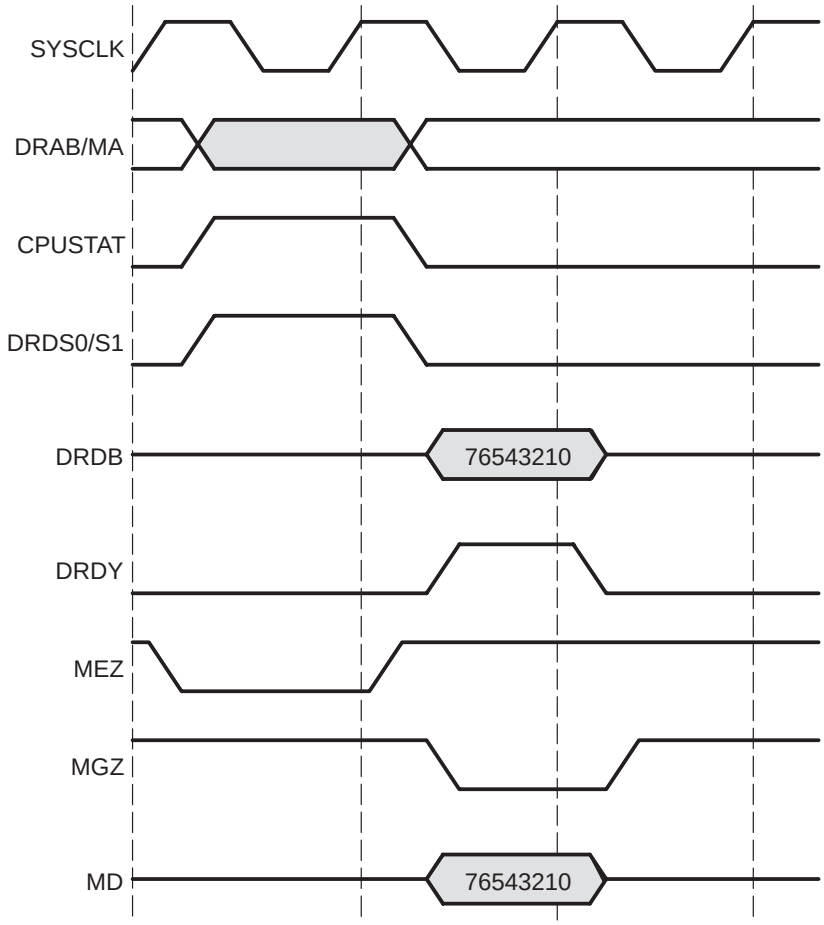
Figure 2–2 through Figure 2–6 show the timing for CROM operations.

Figure 2–2. CROM Program Read Operation (WSTATE = 0)



Note: PRDB receives all 32 bits from the MD bus. The CROM wrapper always performs 32-bit reads, even if the CPU requests a 16-bit read.

Figure 2–3. CROM Data Read Operation (WSTATE = 0)



Note: DRDB receives all 32 bits from the MD bus. The CROM wrapper always performs 32-bit reads, even if the CPU requests a 16-bit read.

Figure 2–4. CROM Simultaneous Program Read and Data Read Operations (WSTATE = 0)

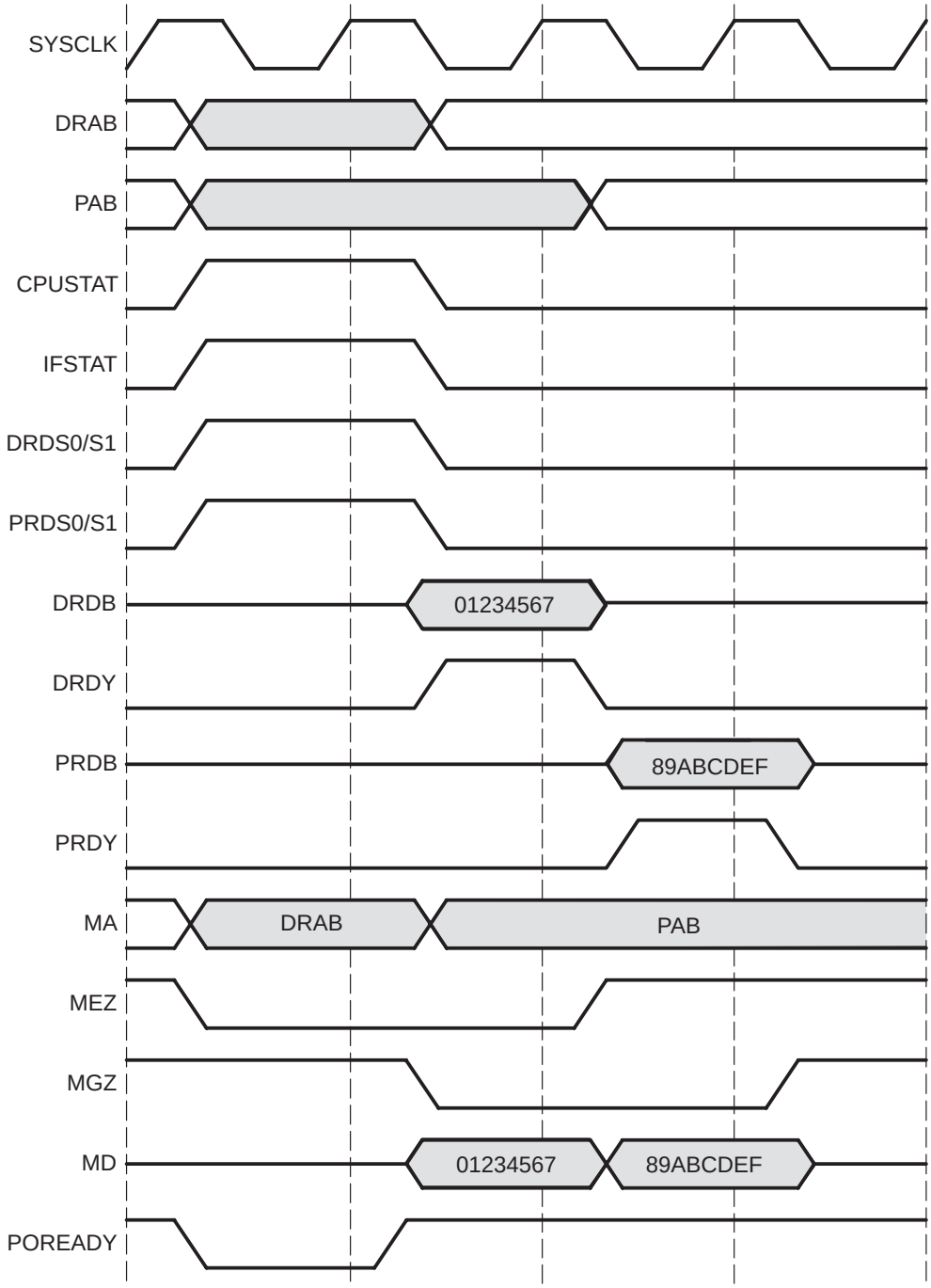


Figure 2–5. CROM Program Read Operation (WSTATE = 1)

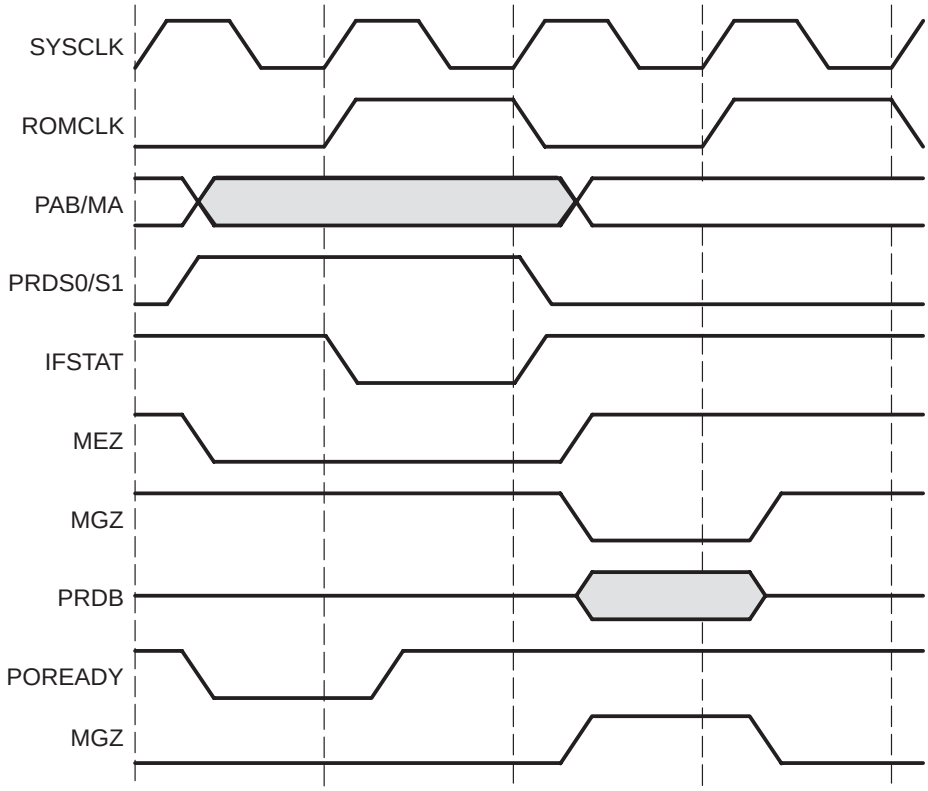
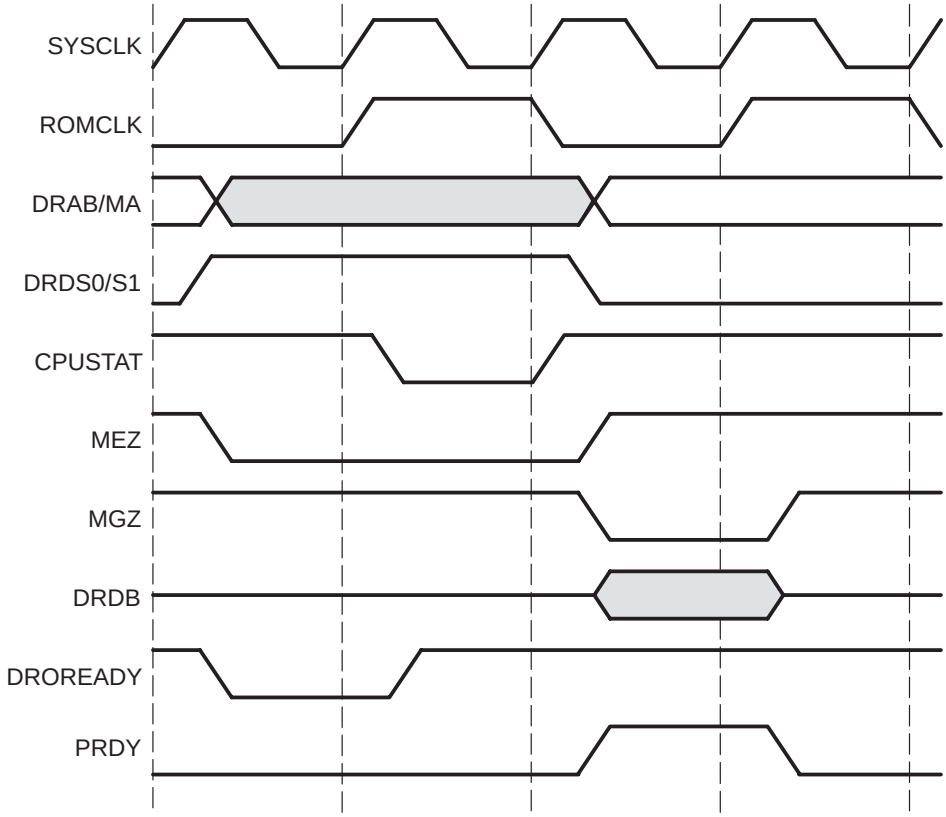


Figure 2–6. CROM Data Read Operation (WSTATE = 1)



2.3 Single-Access RAM (SARAM)

A generic RAM wrapper interfaces with the ’C2700B0 core and the TI ACE single-access RAM (SARAM) cores. This wrapper supports 512, 1K, 2K, 4K, 8K, and 16K 32-bit-wide longwords of SARAM.

The ’C2700B0 core must have at least 256 × 16 words in each of two SARAM blocks called B0 and B1. These blocks are necessary for TI to run different tests on the ’C2700B0 core. For information on B0 and B1 SARAM requirements, see section 2.4 on page 2-21.

For a description of the SARAM wrapper signals, see section A.3 on page A-25.

2.3.1 Memory Core Considerations

There are two different SARAM wrappers. One type of wrapper is used for the MK/MR cores while the other one is used for the MV cores. MK memories are the fastest of the three in terms of memory-access time. MR memories provide larger size options. MV memories are the most compact but are considerably slower in terms of access time. The MV memory wrapper supports wait-state operations, whereas other wrappers do not.

There are also differences between the MK and MR cores. MK memories are bit writable, therefore you can use a single 32-bit wide memory core to interface with the wrapper. MR memories are not bit writable, therefore you must use two parallel banks of 16-bit wide memories to interface with the wrapper.

Although the MK and MR cores have some differences, there is a single wrapper VHDL code and a single netlist that supports both the cores. There are some pins on the wrapper that are specific to the MK cores and others that are specific to the MR cores. The MV memory wrapper has a different VHDL code and netlist from the MK/MR memory wrappers.

Note:

The netlist is synthesized on the assumption that the wrappers are situated close to the memory core and are within 3 millimeters of the core. The bus delays on the ports depend on the location of the wrappers with respect to the memory core while the netlist is being synthesized.

If a different topology need arises, the wrappers are resynthesized using the new topology. For example, if the OREADY path is one of the most critical paths on your cDSP device, you can reduce this path by placing the wrappers very close to the CPU. You must, therefore, put more load on the signals between the wrapper and the memory core and adjust the input and output delays in the synthesis scripts. The synthesis scripts are shipped as part of the ’C2700B0 design kit.

2.3.2 SARAM Configuration Guidelines

Because the ’C2700B0 core can perform 32-bit as well as 16-bit accesses to memory, a memory configuration that supports both core types is used. You can either use two banks of 16-bit wide memory or a single bank of 32-bit wide memory. The 32-bit wide memory can write to 16 upper or lower bits separately. The TSC6000 library contains the MK and MV cores, which are bit writable. This means if you are using MK or MV cores, you can use a single bank of memory configured so that the lower 16 bits and the upper 16 bits are written separately. On the other hand, MR cores are not bit writable. Therefore, to do 16-bit writes using MR cores, you must use two banks of MR cores, each 16 bits wide. One word from each of the two MR cores forms a single 32-bit word.

In either case, the lower 16 bits of the 32-bit word are addressed by an even address, where the upper 16 bits are addressed by an odd address that follows immediately.

The PRDS0, PRDS1, PWDS0, PWDS1, DRDS0, DRDS1, DWDS0, and DWDS1 signals control the alignment and sizes of program reads, program writes, data reads, and data writes (see section A.1.1, *Memory Interface Signals* on page A-2).

The PON pin on the wrapper must be connected to logic 1 if you want to map the memory to program space. If you do not want the block to be mapped in program space, you must connect the PON pin to logic 0. By default, the block is always mapped to data space.

On some of the wrappers, you may encounter a DON signal. If you find a DON signal on a wrapper, tie it high to 1 (as if you are using the ’C2700B0 C compiler and cannot have a memory that is only mapped to program space).

The 12-bit PSTRT[11:0], DRSTRT[11:0], and DWSTRT[11:0] buses on the SARAM wrapper are compared to PAB[21:10], DRAB[21:10], and DWAB[21:10], respectively, from the ’C2700B0 core to determine if the current read access is to the SARAM wrapper. PSTRT, DRSTRT, and DWSTRT are configured according to the size of the SARAM block. If the SARAM block is greater than 512×32 (two 512×16 cores), the lower bits of PSTRT, DRSTRT, and DWSTRT must be connected to the lower bits of PAB[21:10], DRAB[21:10], and DWAB[21:10], respectively.

The address bus MADD[14:1] connects the SARAM wrapper to the two SARAM 16-bit cores. The lowest bit is unconnected because the MCORE-HIGH and MCORELOW enabling signals act as the least significant bit. If the total SARAM memory block is less than 16K $\times$ 32, the upper bits of MADD[14:1] are left unconnected.

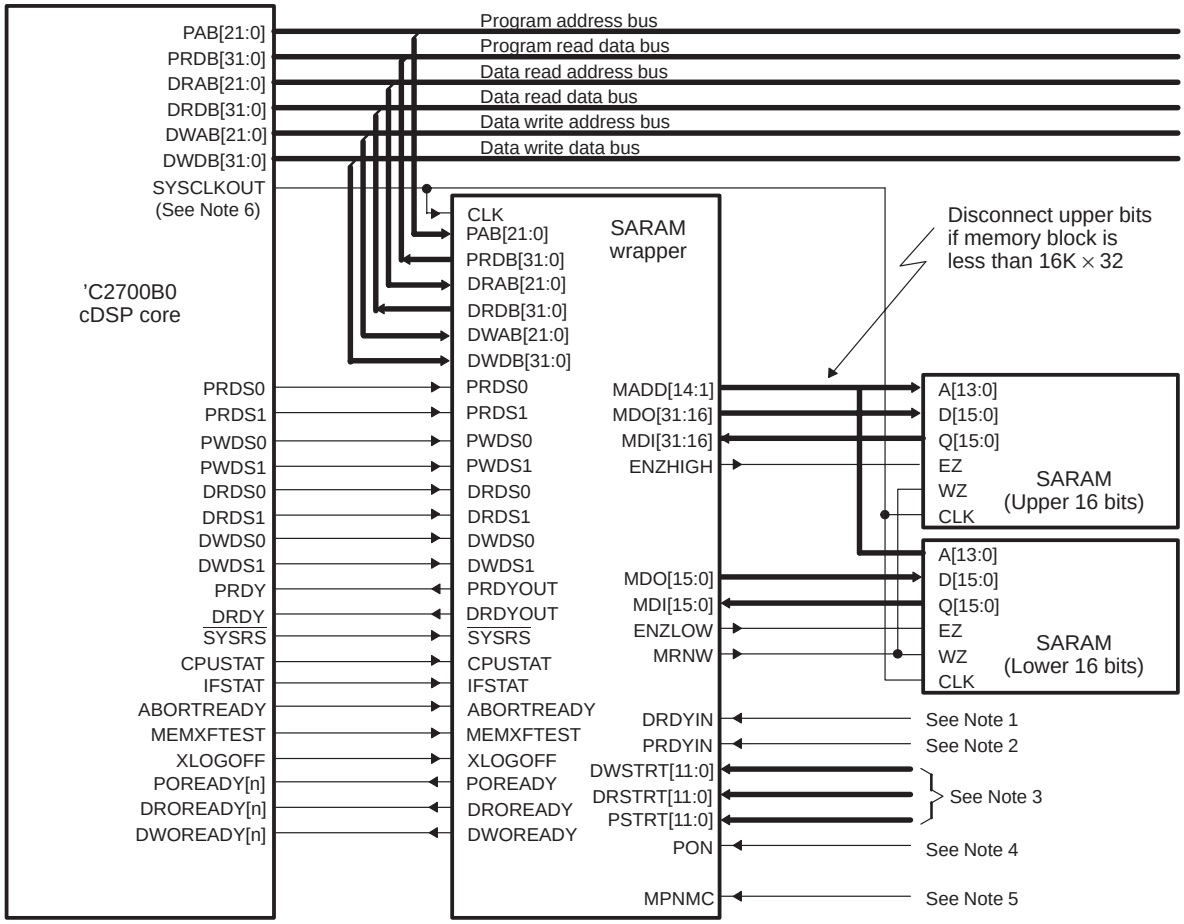
If an SARAM block is less than 512 × 32, then PAB[9:*n*] are unaccounted for on MADD and 2^n-sized mirror images occur to fill the 512 × 32 address range. Table 2–3 summarizes these rules for PSTRT and MADD. These PSTRT concepts apply to data space configurations as well.

Table 2–3. Example SARAM Configurations

SARAM Block Size	Starting Address	PSTRT Configuration	Address Bus to SARAM Cores	Mirrors
128 × 32	01 0000h	PSTRT[11:0] → 00 0001 0000 00	MADD[7:1] → A[6:0]	01 0100h 01 0200h 01 0300h
256 × 32	01 0000h	PSTRT[11:0] → 00 0001 0000 00	MADD[8:1] → A[7:0]	01 0200h
512 × 32	01 0000h	PSTRT[11:0] → 00 0001 0000 00	MADD[9:1] → A[8:0]	–
1K × 32	01 0000h	PSTRT[11:1] → 00 0001 0000 0 PSTRT[0] → PAB[10]	MADD[10:1] → A[9:0]	–
2K × 32	01 0000h	PSTRT[11:2] → 00 0001 0000 PSTRT[1:0] → PAB[11:10]	MADD[11:1] → A[10:0]	–
4K × 32	01 0000h	PSTRT[11:3] → 00 0001 000 PSTRT[2:0] → PAB[12:10]	MADD[12:1] → A[11:0]	–
8K × 32	01 0000h	PSTRT[11:4] → 00 0001 00 PSTRT[3:0] → PAB[13:10]	MADD[13:1] → A[12:0]	–
16K × 32	01 0000h	PSTRT[11:5] → 00 0001 0 PSTRT[4:0] → PAB[14:10]	MADD[14:1] → A[13:0]	–

See Figure 2–7 for a detailed SARAM configuration block diagram.

Figure 2–7. MR SARAM Connections to the T320C2700B0 Core



Notes:

1) Daisy-chained to DRDYOUT of another wrapper for multiple data memory blocks; tied low if SARAM wrapper is first block
2) Daisy-chained to PRDYOUT of another wrapper for multiple program memory blocks; tied low if SARAM wrapper is first block
3) See section 2.3.2, *SARAM Configuration Guidelines*, for PSTRT, DRSTRT, and DWSTRT connection rules.
4) See section 2.3.2, *SARAM Configuration Guidelines*, for PON connection rules.
5) From ASIC logic (if low, SARAM is enabled; if high, SARAM is disabled); ignore when MEMXFTEST is high
6) SYSCLKOUT is not connected directly to CLK on the SARAM wrapper or memory cores, SYSCLKOUT must go through a clock tree synthesis (CTS) buffer.
7) See Figure 5–2 on page 5-8 for connection of scan chain signals: SCEN, SCIN, and SCOUT.

2.3.3 SARAM Timing Diagrams

Figure 2–8 through Figure 2–12 show the timing for SARAM operations.

Figure 2–8. SARAM 16-Bit Even Address Program Read Operation (WSTATE = 0)

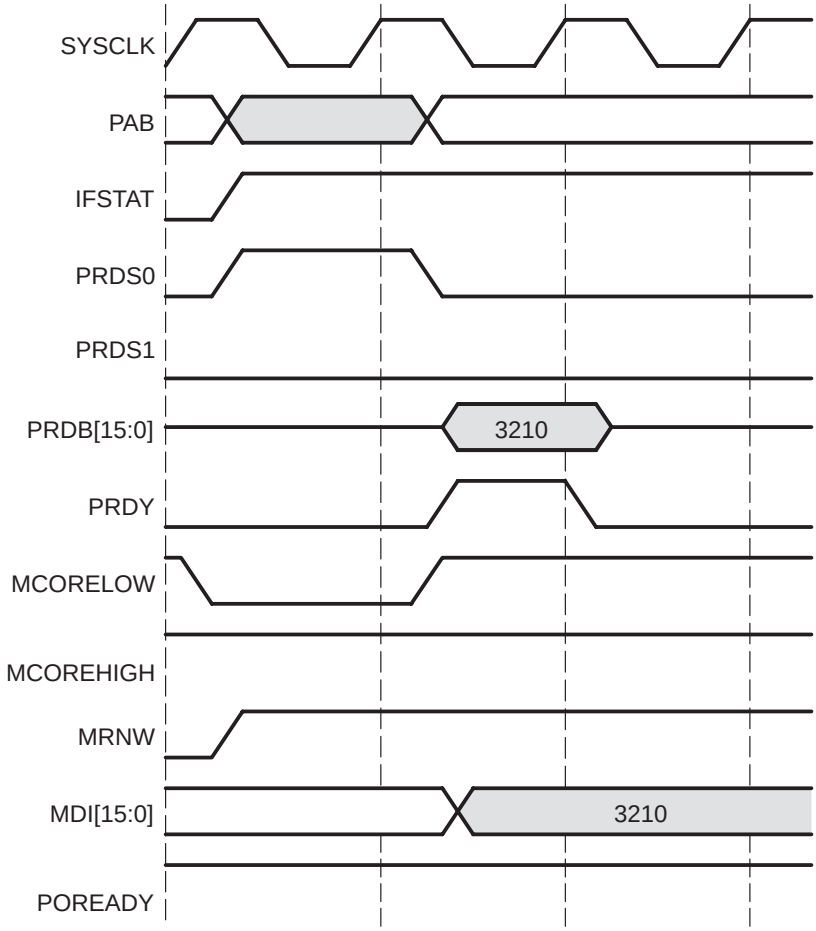


Figure 2–9. SARAM 16-Bit Odd Address Program Read Operation (WSTATE = 0)

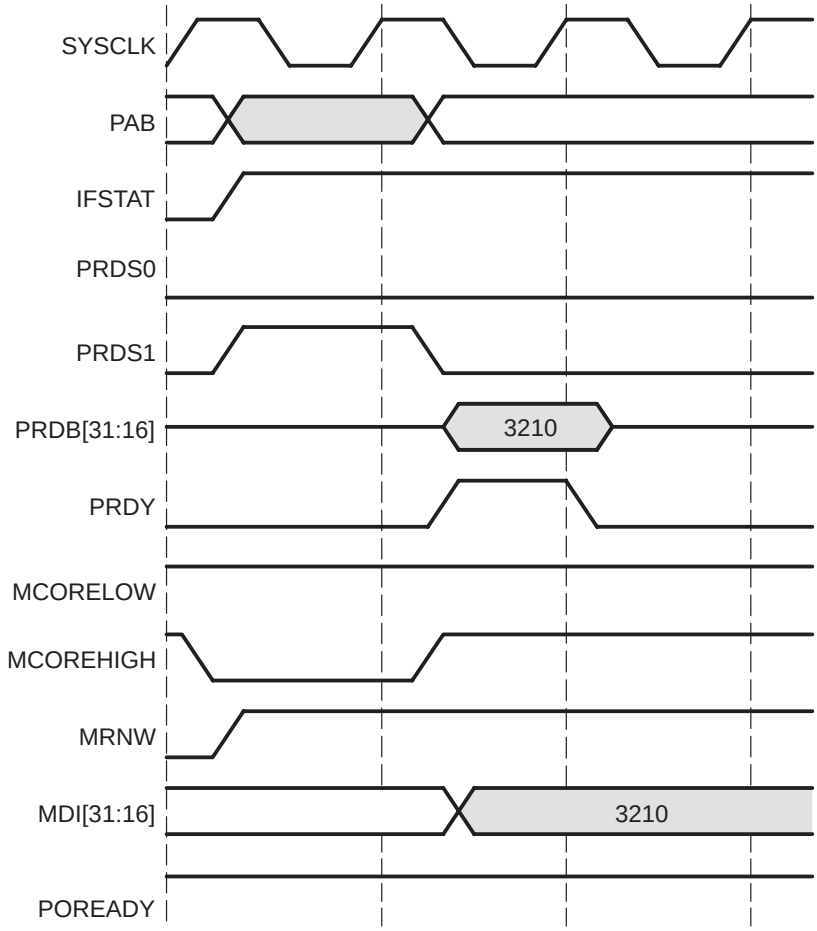


Figure 2–10. SARAM 16-Bit Odd Address Program Write Operation (WSTATE = 0)

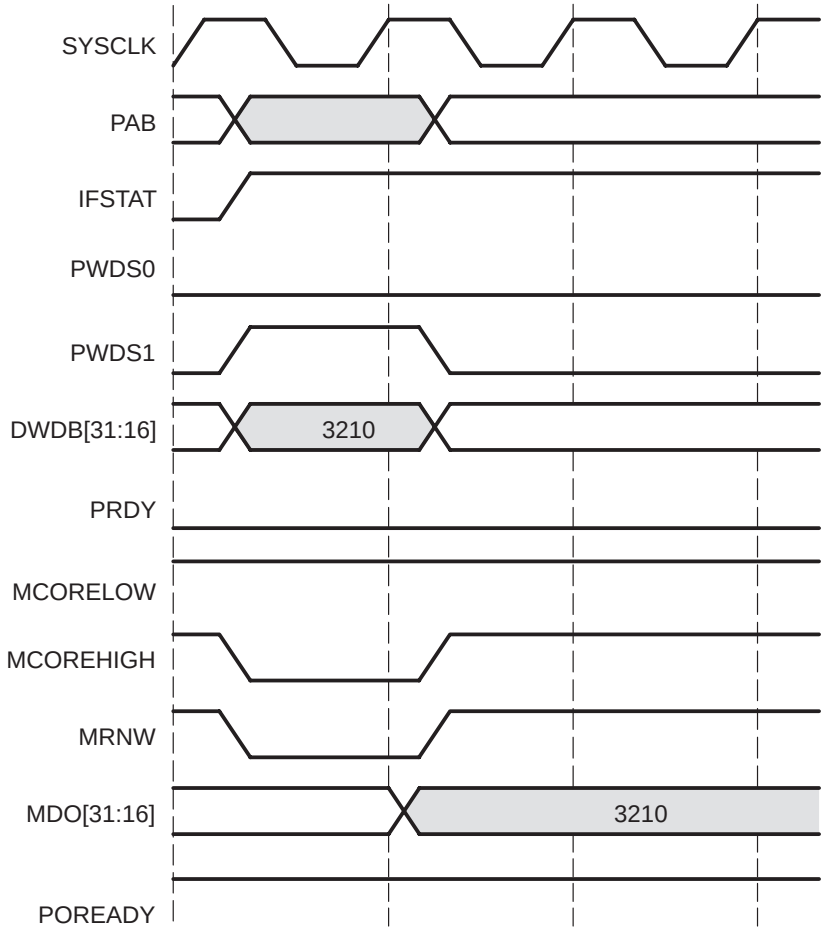


Figure 2–11. SARAM 16-Bit Even Address Data Read Operation (WSTATE = 0)

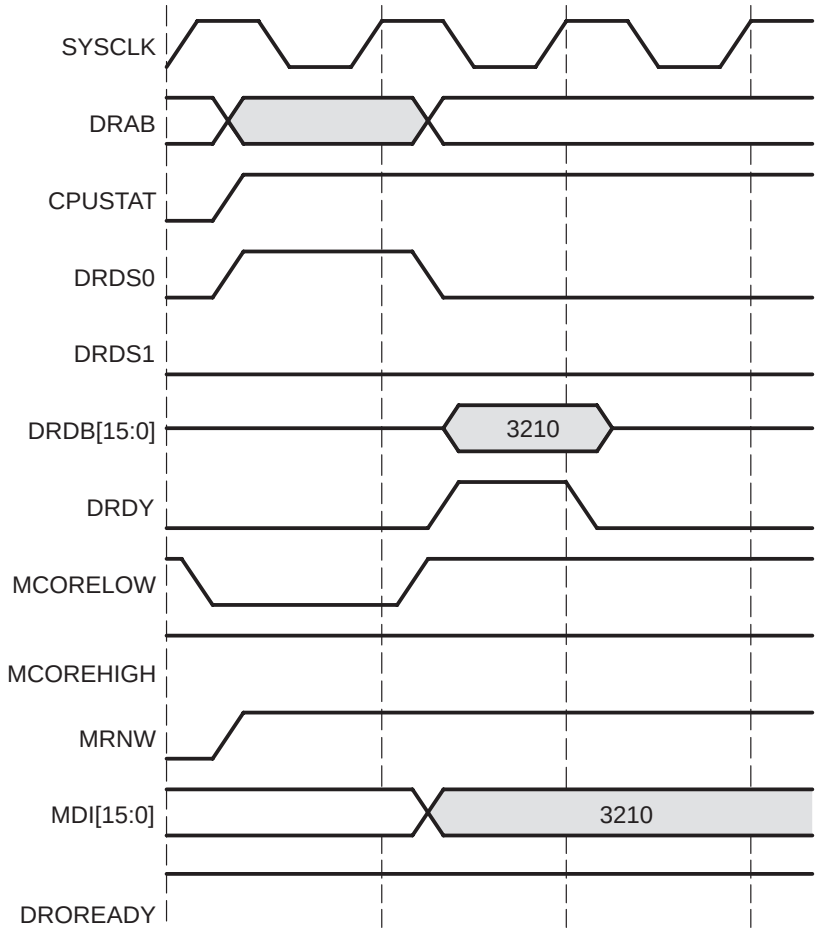
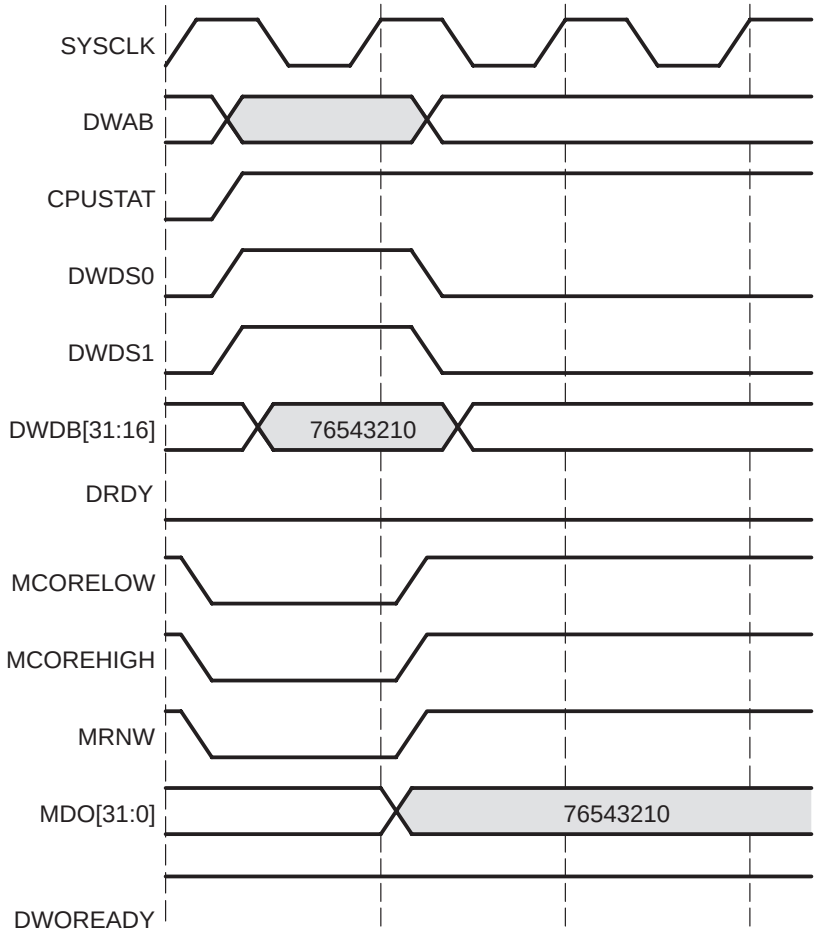


Figure 2–12. SARAM 32-Bit Data Write Operation (WSTATE = 0)



All 32-bit operations are aligned to even addresses.

2.4 B0 and B1 SARAM

The ’C2700B0 core must have at least 256 × 16 words in each of the two SARAM blocks called B0 and B1. These blocks are necessary for TI to run different tests on the ’C2700B0 core. Memory block B0 is mapped to both program and data space in the ’C2700B0 CPU. Memory block B1 is mapped only to data space.

The ’C2700B0 core can be tested with block sizes that are greater than 256 × 16 words of B0 and 256 × 16 words of B1. However, with larger sizes, only specific combinations of B0 and B1 can be emulated on the system emulation device, TMS320C2700-E1. These size combinations are:

- B0 size of 256, 512, or 1K words

  B0 is mapped to program space starting at address 0 and to data space starting at address 400h.

- B1 size of 256, 512, or 1K words

  B1 is mapped to data space at various addresses. B1 of size 256 decodes at multiple addresses (mirroring). These data space addresses are 0h–FFh, 100h–1FFh, 200h–2FFh, and 300h–3FFh. B1 of size 512 decodes at addresses 0h–1FFh and 200h–3FFh. B1 of size 1K decodes at address 0h–3FFh.

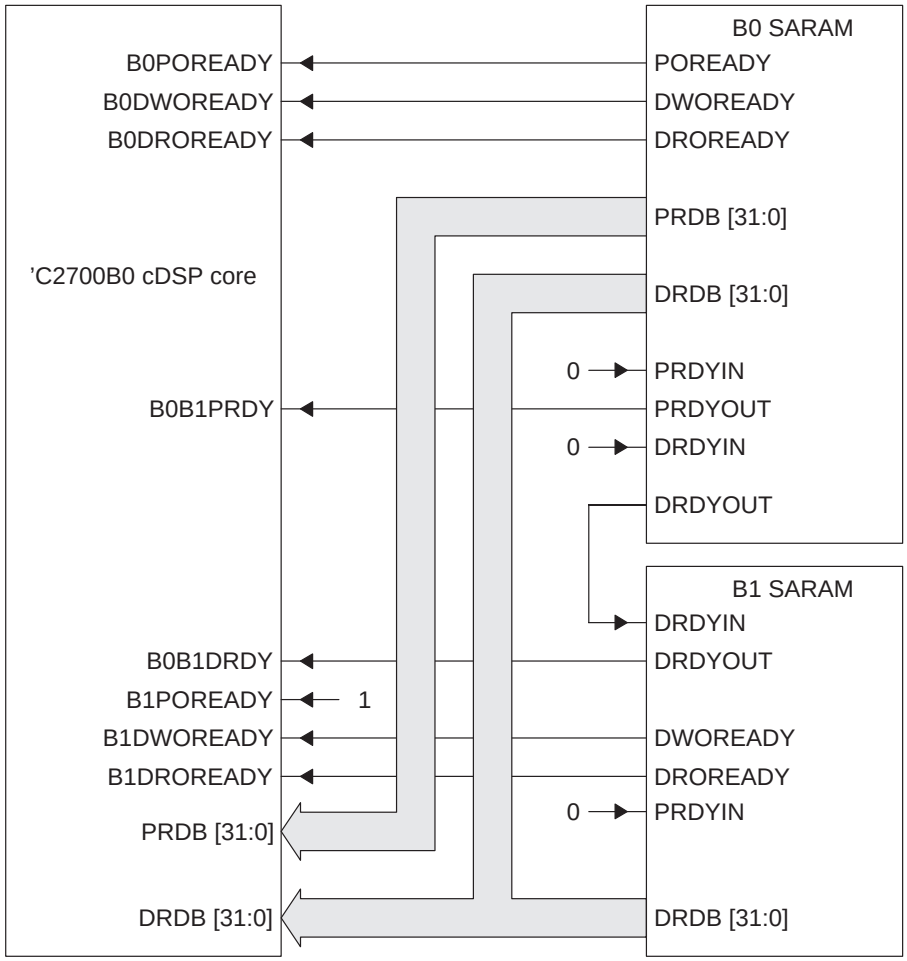
To allow TI to run tests on the ’C2700B0 core, you must connect the B0 and B1 memory blocks to the B0/B1-specific OREADY, PRDY, and DRDY signals as shown in Figure 2–13. These B0 and B1 connections allow the core to disassociate from the surrounding logic, thereby allowing access to B0 and B1 without interference from the user logic.

For descriptions of the OREADY, PRDY, and DRDY core input and memory wrapper signals, as well as other memory signals, see section A.1.1, *Memory Interface Signals*, on page A-2.

Note:

OREADY refers to all memory interface ready signals. These include the DROREADY[13:0], DWOREADY[13:0], and POREADY[13:0] signals, as well as those used by the B0 and B1 memory blocks.

Figure 2–13. B0 and B1 SARAM Connections to the TMS320C2700B0 Core



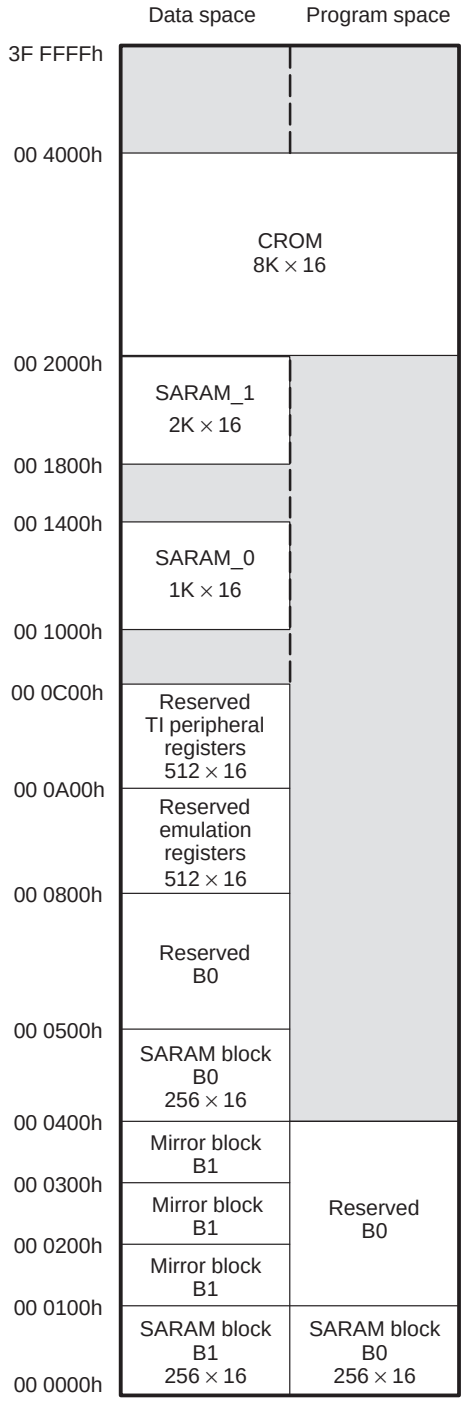
Note: Not all connections are shown. See Figure 2–7 on page 2-15 for a complete connection diagram.

2.5 Example Memory Configuration

This section provides a memory configuration example for a CROM memory map and SARAM that is mapped to both program and data space (see Figure 2–14).

The memory configuration example in this section shows only the unique connections for the memory configuration in Example 2–1 on page 2-25. For complete system connections, see section 2.2.1 on page 2-4 and section 2.3.2 on page 2-13, as well as Appendix A for detailed signal descriptions.

Figure 2–14. Memory Configuration Example Memory Map for CROM and SARAM Mapped to Both Program and Data Space



Example 2–1. Memory Configuration for CROM and SARAM Mapped to Both Program and Data Space

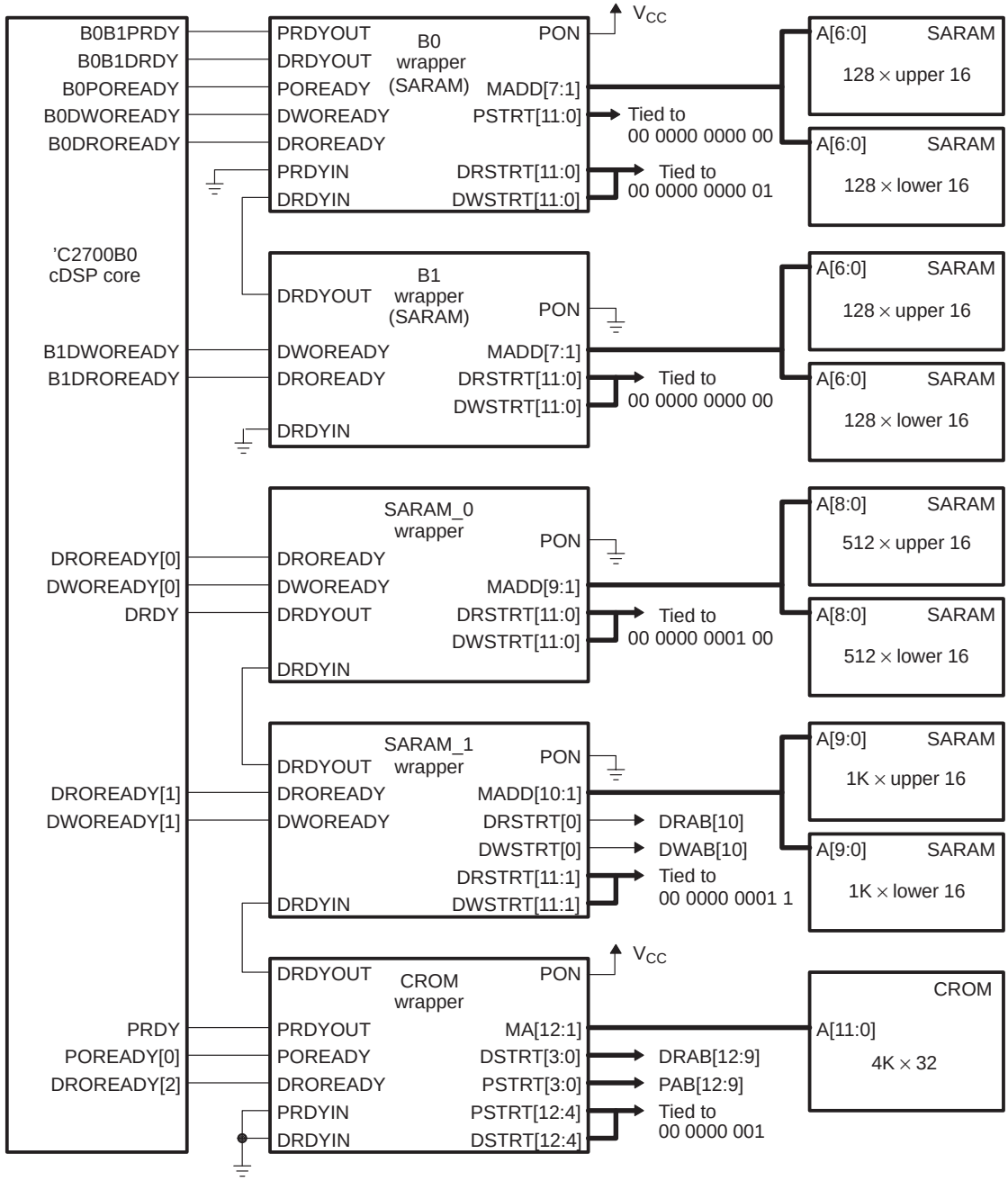
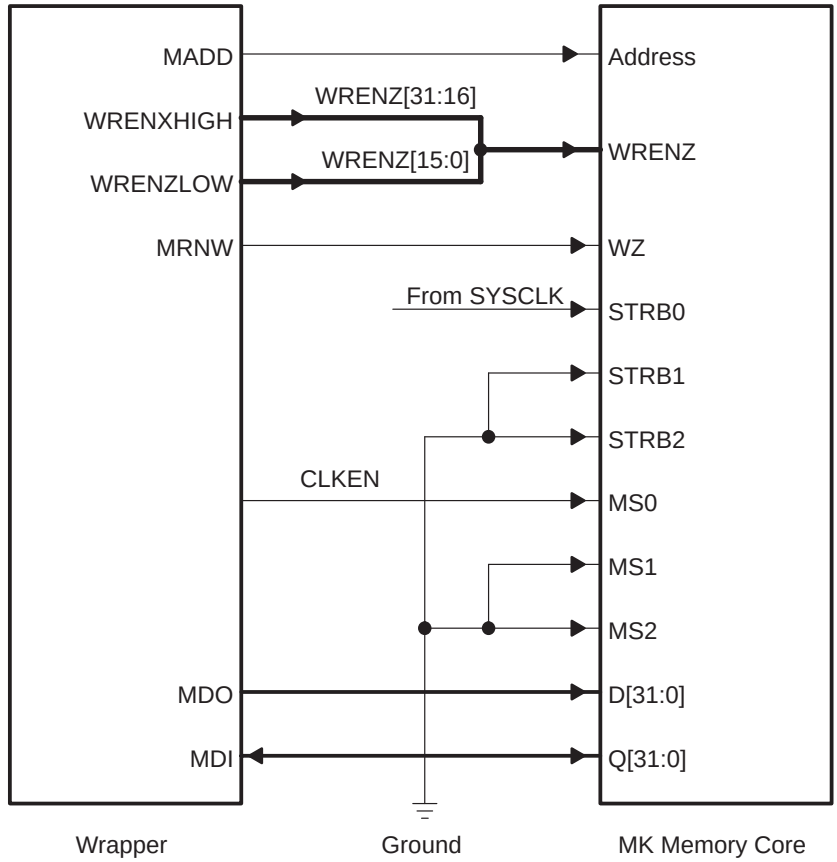


Figure 2–15. Connections Between Wrapper and MK Memory



Chapter 3

External Interface (XINTF)

The external interface (XINTF) is the primary strobe-based interface for standard asynchronous memories and for peripherals originally designed for the T320C2xLP and TMS320C5x devices.

The T320C2700B0 core does not have a dedicated bus for peripheral operations. Instead, separate blocks of TI-provided logic (such as the XINTF) interface I/O blocks to the ’C2700B0 memory buses.

Topic **Page**

3.1 Overview

The XINTF is a strobe-based bus that multiplexes the separate program, data read, and data write buses from the ’C2700B0 core memory bus into a 22-bit address bus and a 16-bit data bus, XA[21:0] and XD[15:0]. The XINTF can support 32-bit, 16-bit, and 8-bit data reads and writes to external memories or peripherals. It also generates wait states to slow external memory accesses.

The XINTF maps over 150 signals of the ’C2700B0 memory protocol to a smaller bus. This bus is intended primarily for interfacing to off-chip components, but it is also available to ASIC designers who prefer strobe-based controls. The XINTF provides:

- A strobe-based interface for attachment of off-chip memories with minimal glue logic
- Supports program tracing through the visibility mode
- Also can support direct memory access (DMA) and cache data transfers that share the external buses
- Adjustable strobe placement for different external peripheral needs and for optimizing memory access time
- Commonality with the logic bus interface used in the T320C2xLP and TMS320C5x cores
- A standard 16-bit data bus with optional extensions to 32 bits
- Adaption of 16-bit peripherals to memory buses by issuing two accesses for each 32-bit memory bus access
- Support for byte operations

The XINTF also filters the external interrupt signals, $\overline{\text{XINT1}}$–$\overline{\text{XINT14}}$ and $\overline{\text{XNMI}}$. Each of the interrupt signals goes through an interrupt filter block that you can configure separately in one of three ways:

- Feed the interrupt directly to the core (the input must be synchronous to the XCLKOUT signal; the input is level sensitive)
- Synchronize the input and level sensitivity (this adds propagation delay but allows for an asynchronous input)
- Synchronize the input and edge sensitivity (this mode generates a single pulse into the interrupt line)

The XINTF is included in the design kit in RTL–VHDL format. For a description of the XINTF signals, see section A.4 on page A-29.

3.2 XINTF Functional Blocks

XINTF is partitioned into 13 functional blocks, as shown in Figure 3–1:

Figure 3–1. External Interface (XINTF) Internal Block Diagram

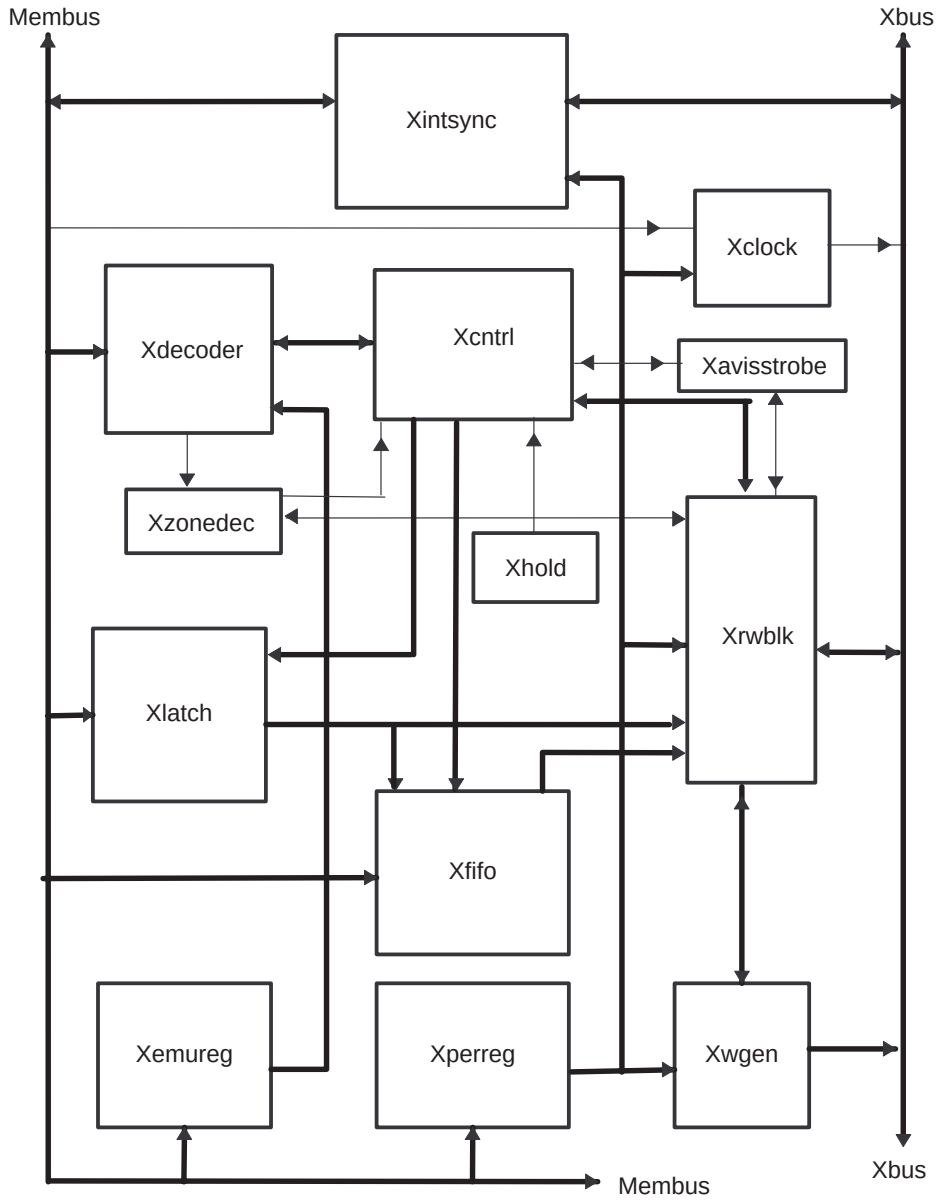
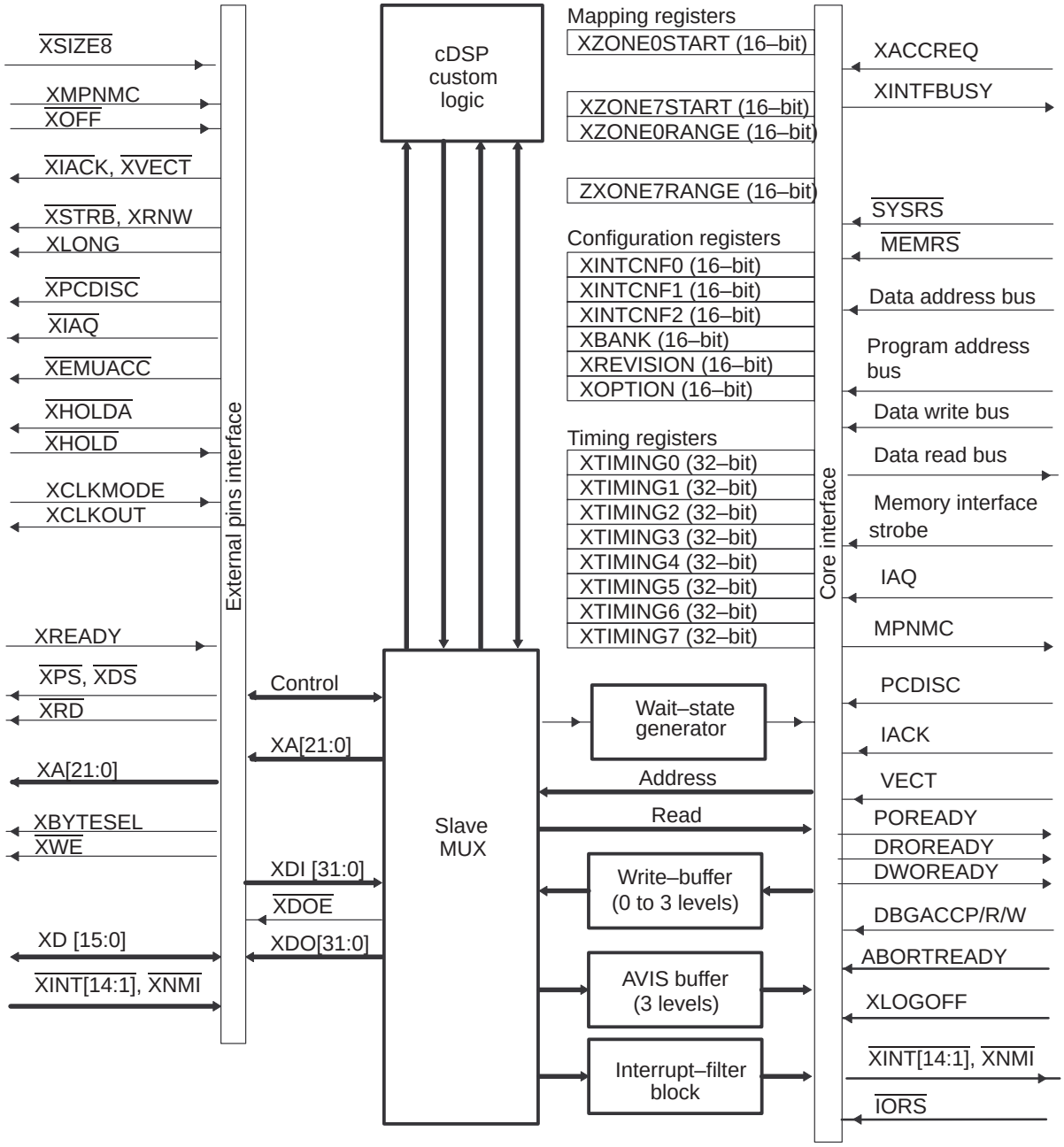


Figure 3–2. External Interface (XINTF) Signals and Register



3.2.1 Xdecoder

The Xdecoder block performs these functions:

- Decodes addresses
- Checks for validity of data read, data write, program read and/or program write and passes valid requests to the FSM logic block
- Checks for interrupt acknowledge from the CPU as well as discontinuity signals and passes the information to the xcntrl block.

The upper 10 bits of the appropriate bus (PAB/DRAB/DWAB, depending on the request) are compared with the start and mask registers of each of the eight zones.

3.2.2 Xclock

The Xclock logic generates the external interface clockout signal, XCLKOUT, which can be configured in two modes:

- Mode 0: xclkout has the same frequency as the CPU clock
- Mode 1: xclkout's frequency is that of the CPU clock divided by 2. The divide by 2 clock is reset on the first high to low transition of $\overline{\text{SYSRS}}$.

3.2.3 Xintsync

The Xintsync is an interrupt synchronizer/filter block. The 14 maskable interrupts and one nonmaskable interrupt to the CPU go through this block. You can configure these interrupts in four modes:

- Mode 00b: an interrupt passed directly to the core. In this mode, interrupts must be synchronized.
- Mode 01b: an interrupt is internally synchronized and is level sensitive
- Mode 10b: an interrupt is internally synchronized and is edge sensitive. In this mode, a trailing edge-sensitive interrupt (a low pulse for one cycle) is generated.
- Mode 11b: an interrupt is not recognized by the XINTF

All interrupts to the XINTF must be held low for at least two clock cycles (the Xintsync block does not check for this condition)

3.2.4 Xemureg

The Xemureg (emulation register) block contains the start and range registers of each of the eight zones. All these registers are 16 bits wide. Only the lower

four bits of DRAB/DWAB are used to access these registers, which are mapped from 980h onwards. The upper 18 bits are used to check on whether the registers that are being accessed are actually 980h onward. Reading from and writing to these registers is similar to typical data read/write operations but you must assert the DRENABLE or the DWENABLE signal while accessing these registers. The DRENABLE and DWENABLE signals come from the emulation logic. See section 3.3, *Remapping External Interface in Memory*, on page 3-9 for more information.

3.2.5 Xperreg

The Xperreg (peripheral register) block contains all the XINTF configuration and control registers. Reading from and writing to these registers is similar to typical data read/write operations. See section 3.4, *External Interface (XINTF) Registers*, on page 3-13 for more information.

3.2.6 Xwgen

The Xwgen block is a wait-state generator. This block generates strobe, address drive, data drive, and data sample signals, and also determines when the current request is over through an internal signal.

This block first decodes the address of requests that were queued in the XINTF. Decoding of requests that were not queued is handled by the xzonedec block.

By decoding the address, this block determines in what zone the current request is. The block then generates strobe and other drive signals, using the count values defined in the corresponding timing zone register. See section 3.4.1, *XINTF Timing Registers (XTIMING)*, on page 3-15 for more information.

3.2.7 Xcntrl

The Xcntrl (control) block controls all the read and write operations and also arbitrates requests. This block is functionally analogous to a finite state machine (FSM). As soon as the request comes from the CPU, this block arbitrates requests and determines what request to service immediately and what requests remain pending. All pending requests are arbitrated and put in a buffer in order. These pending requests are serviced one by one when the current request is over. This technique is easy to implement, is faster, and occupies less area than usual finite state machine (FSM)-based logic.

3.2.8 Xrwblk

The Xrwblk (read/write) block is the interface logic between the Xcntrl block and Xbus. This block does read/write accesses that may have come from the

CPU or or may have been dequeued from the Xfifo. This block translates these accesses into multiple accesses on the external bus as in the case of a 32-bit access to an 8-bit peripheral, a 16-bit access to an 8-bit peripheral, or a 32-bit access to a 16-bit peripheral. This block generates the necessary signals for long accesses and also generates the Xbus strobe signals $\overline{XWE}$, XRNW, $\overline{XPS}$, $\overline{XDS}$, PRDY, and DRDY. This block also generates the $\overline{XPCDISC}$ and $\overline{XVECT}$ visibility mode strobes.

3.2.9 Xfifo

The Xfifo block is a fully synchronous FIFO and enables independent write and read operations. When the FIFO is full, any further write operations are delayed and only read operations can be performed. Similarly, when the FIFO is empty, only write operations can be performed. The Xfifo block buffers all pending requests. The maximum depth of this buffer is 8 and the width is 71. The 22-bit address, 32-bit data, eight data-select signals and XIACK are stored in the buffer along with information on whether the access is an address visibility access and whether it came during a DMA or cache access. The FIFO also accommodates the 3-deep AVIS access buffer. The FIFOOUT bus goes to the Xrwblk block that decodes this FIFOOUT signal and the corresponding operation is performed. The depth of the write buffer in the FIFO is programmable and is configured by the write buffer depth bits 1–0 of XINTCNF2 in the Xperreg block. The depth of the FIFO can be changed only when the FIFO is empty. The write buffer level bits 7–6 of XINTCNF2 can be used to monitor if the FIFO is empty. See section 3.4.2, *XINTF Configuration Registers (XINTCNF)*, on page 3-18 for more information.

3.2.10 Xlatch

The Xlatch block latches current pending requests. These requests are later transferred into the FIFO (Xfifo) in the order determined in the Xcntrl block. This block also sends address/data selects directly to the Xrwblk block when required.

3.2.11 Xzonedec

The Xzonedec block extracts strobe timing and other strobe-related information for accesses that are not queued by the FSM. This block was introduced because the Xwgen block could not do the job quickly enough.

3.2.12 Xavis strobe

Just as the Xwgen block generates the strobes and address drive signals for read and write requests from the XINTF space, this block generates corresponding signals for AVIS requests.

3.2.13 Xhold

This block is used to test the DMA function of the XINTF and it can be programmed to give a DMA request for this purpose.

3.3 Remapping External Interface (XINTF) in Memory

Each of the eight zones can be remapped in memory, just like memory blocks. Because a zone does not have a fixed size, both a beginning and a range must be specified. The memory-map configuration registers are listed in Table 3–1.

To calculate the zone start values, first choose the starting address, making sure it is aligned to a 4K-word boundary and mask out the lower 12 bits. To increase the size of a zone above the default 4K size, use the range register to reduce the number of address bits that are used for zone decode.

Table 3–1. External Interface (XINTF) Memory-Map Configuration Registers

Name	Description	Address	Figure
XZONE0START	XINTF Zone 0 program and data space starting address	0x000980h	Figure 3–3
XZONE1START	XINTF Zone 1 program and data space starting address	0x000981h	Figure 3–3
XZONE2START	XINTF Zone 2 program and data space starting address	0x000982h	Figure 3–3
XZONE3START	XINTF Zone 3 program and data space starting address	0x000983h	Figure 3–3
XZONE4START	XINTF Zone 4 program and data space starting address	0x000984h	Figure 3–3
XZONE5START	XINTF Zone 5 program and data space starting address	0x000985h	Figure 3–3
XZONE6START	XINTF Zone 6 program and data space starting address	0x000986h	Figure 3–3
XZONE7START	XINTF Zone 7 program and data space starting address	0x000987h	Figure 3–3
XZONE0RANGE	XINTF Zone 0 program and data space range mask	0x000988h	Figure 3–4
XZONE1RANGE	XINTF Zone 1 program and data space range mask	0x000989h	Figure 3–4
XZONE2RANGE	XINTF Zone 2 program and data space range mask	0x00098Ah	Figure 3–4
XZONE3RANGE	XINTF Zone 3 program and data space range mask	0x00098Bh	Figure 3–4
XZONE4RANGE	XINTF Zone 4 program and data space range mask	0x00098Ch	Figure 3–4
XZONE5RANGE	XINTF Zone 5 program and data space range mask	0x00098Dh	Figure 3–4
XZONE6RANGE	XINTF Zone 6 program and data space range mask	0x00098Eh	Figure 3–4
XZONE7RANGE	XINTF Zone 7 program and data space range mask	0x00098Fh	Figure 3–4

The register banks are mapped in the reserved emulation register space. They are initialized to default values on a reset by the $\overline{\text{MEMRS}}$ signal (not the $\overline{\text{SYSRS}}$ signal). They can be accessed only by the user software by executing the EALLOW instruction. When you have finished accessing these registers, the EDIS instruction should be executed. These registers are also accessible by way of the JTAG scan port.

For emulation devices, these registers are necessary for configuring the address map of the XINTF such that it matches the target system. For target devices, the XINTF address can be hard-wired so you do not need to program the registers. However, it may be useful to have the hard-wired address be readable so that you can determine the XINTF address space.

Range sizes can only be in binary increments starting from 4K words, all the way up to 2M. The start address must always be set to a multiple of the block size. If, for example, a range size of 16K is chosen, the start address must be set to 16K, 32K, 48K, or 64K and so on. Each zone is mapped to both program and data space simultaneously. Two bits in the start registers (PON and DON) can enable each zone in program or data space separately.

Figure 3–3 shows the bit definitions for the XZONE(0–7)START registers and Figure 3–4 shows the bit definitions for the XZONE(0–7)RANGE registers.

Figure 3–3. XZONE (0–7) START Register Bit Definitions

Register Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bus Address Line	21	20	19	18	17	16	15	14	13	12	X	X	X	X	DON	PON

Notes:
1) X = don't care. Read as 0, writes ignored
2) PON = 1, zone mapped to program space
3) DON = 1, zone mapped to data space
4) Smallest start address increment is 4K words
5) Start address must always be a multiple of the range block size
6) The XMPNMC input signal or the MPNMC bit in the XINTCNF2 register enable Zone 7 if set high (microprocessor mode). If set low, Zone 7 is disabled (microcomputer mode). If Zone 7 is disabled by the MPNMC bit, the state of PON and DON bits is ignored.
7) The start registers are reset to default values by the memory reset signal only ($\overline{\text{MEMRS}}$).

Figure 3–4. XZONE (0–7) RANGE Register Bit Definitions

Register Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Address > Range	21	20	19	18	17	16	15	14	13	12	X	X	X	X	X	X
2M	0	1	1	1	1	1	1	1	1	1	X	X	X	X	X	X
1M	0	0	1	1	1	1	1	1	1	1	X	X	X	X	X	X
512K	0	0	0	1	1	1	1	1	1	1	X	X	X	X	X	X
256K	0	0	0	0	1	1	1	1	1	1	X	X	X	X	X	X
128K	0	0	0	0	0	1	1	1	1	1	X	X	X	X	X	X
64K	0	0	0	0	0	0	1	1	1	1	X	X	X	X	X	X
32K	0	0	0	0	0	0	0	1	1	1	X	X	X	X	X	X
16K	0	0	0	0	0	0	0	0	1	1	X	X	X	X	X	X
8K	0	0	0	0	0	0	0	0	0	1	X	X	X	X	X	X
4K	0	0	0	0	0	0	0	0	0	0	X	X	X	X	X	X

Notes:
1) X = don't care. Read as 0, writes ignored
2) Values not listed in this table are invalid and will yield unpredictable results.
3) The above register is reset to default values by the memory reset signal only ($\overline{\text{MEMRS}}$).

3.3.1 Configuring the Default Start and Range Values

At start up, each zone of the XZONE(0–7)START and XZONE(0–7)RANGE registers is loaded with default values that are determined by tie-offs at the XINTF boundary. This arrangement has the advantage that once you have decided what the size and location of the zones should be, you can make the appropriate tie-offs and the XINTF will always start up with the design specific zone mapping. The default mapping of a zone in either program space or data space or both is also configurable, since the last 2 bits of the START tie-offs are for the DON and PON bits.

The anatomy of the 12-bit XZONE(0–7)START register tie-off is as follows:

Register Bit	11	10	9	8	7	6	5	4	3	2	1	0
											D	P
	<–				Upper 10 bits of the start address					–>	O	O
											N	N

There are eight such tie-offs, one for each of the start registers.

The anatomy of the 10-bit XZONE(0–7)RANGE register tie-off is as follows:

Register Bit	9	8	7	6	5	4	3	2	1	0
	<–				The 10-bit mask value					–>

There are eight such tie-offs, one for each of the range registers.

3.4 External Interface (XINTF) Registers

The XINTF contains several configuration registers that are read from and written to by the memory bus side of the XINTF. The registers set up the bus timing, specify how interrupts are conditioned, configure the write buffer depth, and other options. Typically, they are initialized by system software during the power-up sequence.

The operation and timing of the XINTF are controlled by the timing and configuration registers, listed in Table 3–2. These registers are mapped in the reserved peripheral register space. They are initialized to default values on a reset by the $\overline{\text{SYSRS}}$ signal (not the $\overline{\text{MEMRS}}$ signal).

Table 3–2. External Interface (XINTF) Timing and Configuration Registers

Name	Description	Address
XTIMING0	XINTF timing register, Zone 0	0x000B20h
XTIMING1	XINTF timing register, Zone 1	0x000B22h
XTIMING2	XINTF timing register, Zone 2	0x000B24h
XTIMING3	XINTF timing register, Zone 3	0x000B26h
XTIMING4	XINTF timing register, Zone 4	0x000B28h
XTIMING5	XINTF timing register, Zone 5	0x000B2Ah
XTIMING6	XINTF timing register, Zone 6	0x000B2Ch
XTIMING7	XINTF timing register, Zone 7	0x000B2Eh
XINTCNF0	XINTF $\overline{\text{XINT1}}$ to $\overline{\text{XINT8}}$ configuration register	0x000B30h
Reserved		0x000B31h
XINTCNF1	XINTF $\overline{\text{XINT9}}$ to $\overline{\text{XINT14}}$ configuration register	0x000B32h
Reserved		0x000B33h
XINTCNF2	XINTF configuration register	0x000B34h
Reserved		0x000B35h to 0x000B37h
XBANK	XINTF bank control register	0x000B38h
Reserved		0x000B39h
XREVISION	XINTF revision register	0x000B3Ah
Reserved		0x000B3Bh
XOPTION	XINTF option register	0x000B3Ch
Reserved		0x000B3Dh to 0x000B3F

3.4.1 XINTF Timing Registers

XINTF signal timing can be tuned to match specific external device requirements, such as:

- ❑ Setup and hold times
- ❑ To strobe signals for avoiding contention or minimizing access time

The timing parameters can be configured individually for different address regions. This allows you to maximize the efficiency of the bus, based on the type of memory or peripheral that you need to access. On the XINTF, the timing parameters can be configured for each of the eight zones, Zone 0 to Zone 7. The XTIMING is shown in Figure 3–5 and described in Table 3–3. On system reset ($\overline{\text{SYSRS}}$), the XTIMING should be set to FFFFh to force the loosest timing on power up. This also enables sampling of XREADY and selects asynchronous mode of the XREADY operation.

The individual timing parameters can be programmed into the XTIMING registers as described in Table 3–3. For devices that do not require programmability of such parameters, the timings can be hard wired; however, the hard-wired configuration values should be readable in the timing register fields. PREWRLEAD, PRERDLEAD, and X2TIMING are optional. The implemented options are indicated in the XOPTION register.

Figure 3–5. XINTF Timing Register (XTIMING)

22	21–20	19–18	17	16	15	14
X2TIMING	PRERDLEAD	PREWRLEAD	$\overline{\text{XSIZE32}}$ Mode	$\overline{\text{XSIZE8}}$ Mode	READY MODE	USEREADY
R/W	R/W	R/W	R/W	R/W	R/W	R/W

13–12	11–9	8–7	6–5	4–2	1–0
XRDLEAD	XRDACTIVE	XRDTRAIL	XWRLEAD	XWRACTIVE	XWRTRAIL
R/W	R/W	R/W	R/W	R/W	R/W

Note: R = Read access; W = Write access

Table 3–3. XINTF Timing Register (XTIMING) Field Descriptions

Bits	Name	Description
22	X2TIMING	This bit specifies the scaling factor of the LEAD, ACTIVE, TRAIL values in the individual timing registers. If this bit is 0, the values are scaled 1:1. If this bit is 1, the values are scaled 2:1 (doubled). On system reset ($\overline{\text{SYSRS}}$), this bit is set to 1 (maximum cycles). For 100+ MHz systems, you may need to extend the wait states to be able to talk to slow memory devices.
21–20	PRERDLEAD	This 2-bit field specifies the number of SYSCLKOUT cycles, from 0, 1, 2, 3 (if X2TIMING bit is 0) or 0, 2, 4, 6 (if X2TIMING bit is 1), to add to the current number of lead cycles, when a read cycle follows any write cycle. On system reset ($\overline{\text{SYSRS}}$), these bits are set to 1 (maximum cycles).
19–18	PREWRLEAD	This 2-bit field specifies the number of SYSCLKOUT cycles, from 0 1, 2, 3 (if X2TIMING bit is 0) or 0, 2, 4, 6 (if X2TIMING bit is 1) to add to the current number of lead cycles, when a write cycle follows any read cycle. On system reset ($\overline{\text{SYSRS}}$) these bits are set to 1 (maximum cycles).
17	$\overline{\text{XSIZE32}}$ Mode	When set to 0, this bit enables the specified zone to support 32-bit wide memories. When set to a 1, the specified zone supports 8-bit or 16-bit memories based on the state of the $\overline{\text{XSIZE8}}$ bit. The state of this bit on a system reset ($\overline{\text{SYSRS}}$) is the value of the $\overline{\text{XSIZE32}}$ input signal. The $\overline{\text{XSIZE32}}$ input signal state is ignored after reset.
16	$\overline{\text{XSIZE8}}$ Mode	When set to 0, this bit enables the specified zone to support 8-bit wide memories. When set to 1, the specified zone supports 16-bit or 32-bit memories based on the state of the $\overline{\text{XSIZE32}}$ input signal. The $\overline{\text{XSIZE8}}$ input signal state is ignored after reset. $\overline{\text{XSIZE8}}$ and $\overline{\text{XSIZE32}}$ must never both be set to 0. If both are set to 0, the bus reverts to 16-bit wide mode of operation.
15	READY-MODE	When set, the XREADY input is asynchronous. When cleared, the XREADY input is synchronous.
14	USEREADY	When this bit is set (default), the XREADY signal is used to further extend the active portion of the cycle past the minimum defined by the XRDACTIVE and XWRACTIVE fields. When this bit is cleared, the XREADY signal is ignored.
13–12	XRDLEAD	A 2-bit field that defines the setup time of the read cycle in SYSCLKOUT cycles, from 0, 1, 2, 3 (if X2TIMING bit is 0) or 0, 2, 4, 6 (if X2TIMING bit is 1).
11–9	XRDACTIVE	A 3-bit field that defines the read cycle active period, in SYSCLKOUT cycles, from 0, 1, 2, 3, 4, 5, 6, 7 (if X2TIMING bit is 0) or 0, 2, 4, 6, 8, 10, 12, 14 (if X2TIMING bit is 1). The active period is by default one cycle. The total active period is 1 plus XRDACTIVE value.
8–7	XRDTRAIL	A 2-bit field that defines the read cycle trail period, in SYSCLKOUT cycles, from 0, 1, 2, 3 (if X2TIMING bit is 0) or 0, 2, 4, 6 (if X2TIMING bit is 1).
6–5	XWRLEAD	A 2-bit field that defines the write cycle lead period, in SYSCLKOUT cycles from 0, 1, 2, 3 (if X2TIMING bit is 0) or 0, 2, 4, 6 (if X2TIMING bit is 1).

Table 3–3. XINTF Timing Register (XTIMING) Field Descriptions (Continued)

Bits	Name	Description
4–2	XWRACTIVE	A 3-bit field that defines the write cycle active period, in SYSCLKOUT cycles, from 0, 1, 2, 3, 4, 5, 6 7 (if X2TIMING bit is 0) or 0, 2, 4, 6, 8, 10, 12 ,14 (if X2TIMING bit is 1). The active period is by default one cycle. The total active period is 1 plus the XWRACTIVE value.
1–0	XWRTRAIL	A 2-bit field that defines the write cycle trail period in SYSCLKOUT cycles from 0, 1, 2, 3 (if X2TIMING bit is 0) or 0, 2, 4, 6 (if X2TIMING bit is 1).

The PREWRLEAD and PRERDLEAD parameters may be necessary to tune bus turnaround time when going between very fast read and write accesses. However, if this is not an issue for the targeted system, this feature can be left off. For optional features not implemented, bits must read 0 and writes are ignored.

3.4.2 XINTF Configuration Registers (XINTCNF)

XINTCNF0 (Figure 3–6) and XINTCNF1 (Figure 3–7) determine what conditioning the XINTF does to XINT[14:1] inputs before sending them to the ’C2700B0 core.

For $\overline{\text{XNMI}}$ and each individual $\overline{\text{XINT}}$ input, a 2-bit mode field determines the conditioning for that signal, see Table 3–4.

Figure 3–6. XINTF $\overline{\text{XINT1}}$ to $\overline{\text{XINT8}}$ Configuration Register (XINTCNF0)

15–14	13–12	11–10	9–8	7–6	5–4	3–2	1–0
INT8 mode	INT7 mode	INT6 mode	INT5 mode	INT4 mode	INT3 mode	INT2 mode	INT1 mode
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Note: R = read access; W = write access

Figure 3–7. XINTF $\overline{\text{XINT9}}$ to $\overline{\text{XINT14}}$ Configuration Register (XINTCNF1)

15–12	11–10	9–8	7–6	5–4	3–2	1–0
Spare	INT14 mode	INT13 mode	INT12 mode	INT11 mode	INT10 mode	INT9 mode
X	R/W	R/W	R/W	R/W	R/W	R/W

Note: R = read access; W = write access; X = don’t care

Table 3–4. XINTF Conditioning to $\overline{\text{XINT14}}$–$\overline{\text{XINT1}}$

Mode Field	Conditioning
00	No conditioning
01	Synchronize to processor clock without edge detection (level sensitive). This is the power up value for each field.
10	Synchronize and detect leading edge. The XINTF issues a synchronized 1-processor-cycle pulse for each assertion of the interrupt.
11	Undefined

XINTCNF2 (shown in Figure 3–8 and described in Table 3–5) sets the conditioning of the $\overline{\text{XNMI}}$ input and contains the XCLKOUT mode field, the write buffer depth field, a read-only bit of the XINTF $\overline{\text{XOFF}}$ input signal.

Figure 3–8. XINTF (XINTCNF2) Diagram

15–14	13	12	11	10	9	8
$\overline{\text{XNMI}}$ mode	AVIS mode	XEAVIS mode	$\overline{\text{XHOLDA}}$ signal status	$\overline{\text{XHOLD}}$ signal status	HOLD mode disable	MPNMC mode
R/W	R/W	R	R	R	R/W	R/W

7–6	5	4	3	2	1–0
Write buffer level	Reserved	Reserved	XCLKOUT off	XCLKOUT mode	Write buffer depth
R			R/W	R/W	R/W

Note: R = read access; W = write access; X = don’t care

Table 3–5. XINTF Configuration Register (XINTCNF2) Field Descriptions

Bits	Name	Description
15–14	NMI mode	The NMI mode bits determine the conditioning for the $\overline{\text{XNMI}}$ signal. 00 No conditioning 01 Synchronize to processor clock without edge detection (level sensitive). This is the power up value for each field. 10 Synchronize and detect leading edge. The XINTF issues a synchronized 1-processor-cycle pulse for each assertion of the interrupt. 11 Undefined
13	AVIS mode	When set high, the AVIS mode bit enables the PC discontinuity visibility mode, generating a four-cycle XINTF bus access to export internal PC discontinuities when they occur. When this bit is set low, the PC discontinuity visibility mode is turned off. This bit is set high on a system reset ($\overline{\text{SYSRS}}$).

Table 3–5. XINTF Configuration Register (XINTCNF2) Field Descriptions (Continued)

Bits	Name	Description
12	XEAVIS mode	When set high, this bit enables the export of PC discontinuity information on the XINTF pins, if AVIS mode is set. If AVIS mode is not set, the XEAVIS bit has no effect. If the XEAVIS bit is set low, the PC discontinuity information is not exported on the XINTF bus and all signals stay in an inactive state for the duration of visibility cycles (as if no bus activity is present). The purpose of this function is to reduce electromagnetic interference (EMI) noise on the pins when not tracing visibility information; however, you still want the cycle hit due to the visibility mode to be present. If this bit is set high, exporting of visibility mode information is enabled on a system reset ($\overline{\text{SYSRS}}$). The following table is the truth table for AVIS and XEAVIS bits: AVIS — XEAVIS — Operation 0 — X — Visibility off, export XINTF off 1 — 0 — Visibility on, export XINTF off 1 — 1 — Visibility on, export XINTF on
11	$\overline{\text{XHOLDA}}$ signal status	This bit reflects the current state of the $\overline{\text{XHOLDA}}$ output signal. You can read it to determine if the external interface is granting access to an external device.
10	$\overline{\text{XHOLD}}$ signal status	This bit reflects the current state of the $\overline{\text{XHOLD}}$ input signal. You can read it to determine if an external device is requesting access to the external bus.
9	HOLD mode disable	When low, this bit automatically grants a request to an external device driving the $\overline{\text{XHOLD}}$ input signal low ($\overline{\text{XHOLDA}}$ output signal is driven low when request granted). If this bit is set while $\overline{\text{XHOLD}}$ and $\overline{\text{XHOLDA}}$ are both low (external bus accesses granted) the $\overline{\text{XHOLDA}}$ signal is forced high (at the end of the current cycle) and the external interface is taken out of high-impedance mode. On a system reset ($\overline{\text{SYSRS}}$), this bit is set to zero. If on a system reset the $\overline{\text{XHOLD}}$ signal is active low, the bus and all signal strobes must be in high impedance mode and the $\overline{\text{XHOLDA}}$ signal also driven active low. When HOLD mode is enabled and $\overline{\text{XHOLDA}}$ is active low (external bus grant active) the T320C2700B0 core can still execute code from internal memory. If an access is made to the external interface, a not ready signal is generated and the core is stalled until the $\overline{\text{XHOLD}}$ signal is removed.

Table 3–5. XINTF Configuration Register (XINTCNF2) Field Descriptions (Continued)

Bits	Name	Description
8	MPNMC mode	On reset, this bit reflects the state of the XMPNMC input signal (as sampled by $\overline{\text{SYSRS}}$). The state of this bit is also reflected on the MPNMC output signal from the XINTF block. You can modify the state of this bit by writing a 1 or a 0 to this location. THis is reflected on the MPNMC output signal. This mode also affects Zone 7 decode as follows: MPNMC = 1, microprocessor state (Zone 7 enabled, PON and DON bits active) MPNMC = 0, microcomputer state (Zone 7 disabled, PON and DON bits ignored) The XMPNMC input signal state is ignored after reset.
7–6	Write buffer level	The write buffer level bits indicate the current number of values in the buffer. 00 Empty 01 One value currently in the write buffer (can be 8-, 16-, or 32-bit data) 10 Two values currently in the write buffer (can be 8-, 16-, or 32-bit data) 11 Three values currently in the write buffer (can be 8-, 16-, or 32-bit data)
5	Reserved	
4	Reserved	
3	XCLKOUT off	When this bit is set to 1, the XCLKOUT signal is turned off. This is done for power savings and noise reduction. This bit is set to 0 on a reset ($\overline{\text{SYSRS}}$).
2	XCLKOUT mode	If this bit is set to 1, XCLKOUT is a divide by 2 of the SYSCLKOUT. If this bit is set to 0, XCLKOUT is equal to SYSCLKOUT. All bus timings, irrespective of which mode is enabled, start from the rising edge of XCLKOUT. The default mode of operation on power up and a system reset ($\overline{\text{SYSRS}}$) is determined by the XCLKMODE input signal. If high, it defaults to divide by 2 mode. If low, it defaults to divide by one mode. After being latched by reset, you can modify the mode by writing to these bits. The XCLKMODE input signal state is ignored after reset.
1–0	Write buffer depth	The write buffer depth bits allow the processor to continue execution without waiting for XINTF write accesses to complete. The write buffer depth is selectable as follows: 00 No write buffering. The processor is stalled until XINTF write accesses are complete. This is the power-up value. 01 The XINTF buffers one word. The processor is stalled only if a second XINTF access is accepted before the first finishes. 10 Up to two XINTF write accesses can be buffered

Table 3–5. XINTF Configuration Register (XINTCNF2) Field Descriptions (Continued)

Bits	Name	Description
		11 Up to three XINTF write accesses can be buffered The buffered access can be 8, 16, or 32 bits in length. Order of execution is preserved; for example, writes are performed in the order they were accepted. The processor is stalled on XINTF reads until all pending writes are done and the read access finishes. When the buffer is full the processor is stalled. The write buffer depth bits can be changed; however, it is recommended that the write buffer depth bits be changed only when the buffer is empty (this can be checked by reading the write buffer level bits). Writing to the write buffer depth bits when the level is not 0 can cause unpredictable results.

Figure 3–9. XBANK Configuration Register Diagram

15	14–6	5–3	2:0
EIACK	Spares	Dead cycles	Bank zone number
R/W		R/W	R/W

To assist in dealing with interfacing to fast and slow memories, the XINTF can support a feature where the dead cycles can be added between consecutive cycles that cross a specified address boundary. The address boundary is selectable from any one of fixed zone boundaries as specified by the timing registers (Zone 0 to Zone 7).

The dead cycles are added only on consecutive cycles that cross the boundary, and not within consecutive cycles within the specified zone. The cycles are added on read-to-write, write-to-read, write-to-write cycles irrespective of whether the cycle is a program or data space access.

There is no bit for turning off the inclusion of the dead cycles; however, you can achieve this by setting the number of dead cycles to 000.

Table 3–6. XBANK Configuration Register Field Descriptions

Bits	Name	Description
15	EIACK	Enable IACK operation on XINTF. The IACK operation uses the Zone 0 timing register parameters (XTIMING0). The PON and DON bits in the XZONEOSTART register are ignored. On a system reset ($\overline{\text{SYSRS}}$), this bit is set, hence enabling IACK operation.
14–6	Spares	

Table 3–6. XBANK Configuration Register Field Descriptions (Continued)

Bits	Name	Description
5–3	Dead cycles	These bits specify the number of SYSCLKOUT signals to add between any consecutive cycle that crosses into or out of the specified zone, be it a read or write program or data space. The number of SYSCLKOUT cycles can be 0 to 7. On a system reset ($\overline{\text{SYSRS}}$), this bit is set, enabling IACK operation.
2–0	Bank zone number	These bits specify the XINTF zone for which bank switching is enabled, Zone 0 to Zone 7. On a system reset ($\overline{\text{SYSRS}}$), the value defaults to Zone 7.

3.4.3 Configuration Registers (XREVISION/XOPTION)

To determine which revision of the XINTF is currently being used, a revision register is hard coded into the XINTF at a specified address. The register bits are defined as follows:

Figure 3–10. XREVISION Register

7	6	5	4	3	2	1	0
Revision							
R							

Note: R = Read

Table 3–7. XREVISION Register Field Descriptions

Bits	Name	Description	
15–0	Revision	Current reserved revisions:	
		Version of XINTF	Rev Number
		E1–XINTF (not implemented)	0x0000
		P2–XINTF	0x0001
		Spares	0x0002 to 0xFFFE
		Link to next revision register	0xFFFF

Figure 3–11. XOPTION Configuration Register Diagram

15	14	13	12	11	10	9	8
Link	XMSC mode	DISVECTF mode	C27XDMA bus	Cache support	Security feature	32-bit bus	X2TIMING mode
R	R	R	R	R	R	R	R

7	6	5	4	3	2	1	0
PRELEAD modes	Bank mode	AVIS mode	HOLD ext DMA	XCLKOUT off mode	XCLKOUT divide-by-2 mode	XLOCK	Write buffer
R	R	R	R	R	R	R	R

Note: R = Read

Table 3–8. XOPTION Configuration Register Field Descriptions

Bits	Name	Description
15	Link	If this bit is set, it indicates that there is an extension to this register.
14	XMSC mode	Specifies if the microstate complete output signal mode has been implemented
13	DISVECTF mode	Specifies whether the disable vector fetch input signal mode has been set
12	C27XDMA bus	Specifies whether the C27X DMA bus feature has been implemented
11	Cache support	Specifies whether the cache support feature has been implemented
10	Security feature	Specifies whether the XINTF security feature has been implemented
9	32-bit bus	Specifies whether a full 32-bit bus has been pinned out
8	X2TIMING mode	Specifies whether the 2:1 scaling of lead, active, trail timing values has been implemented
7	Prelead modes	Specifies whether the preread/prewrite mode has been implemented
6	BANK mode	Specifies whether the bank switch protection mode has been implemented
5	AVIS mode	Specifies whether the address visibility mode has been implemented
4	HOLD external DMA	Specifies whether the XHOLD/XHOLDA external DMA function has been implemented

Table 3–8. XOPTION Configuration Register Field Descriptions (Continued)

Bits	Name	Description
3	XCLKOUT off mode	Specifies whether the XCLKOUT off mode has been implemented
2	XCLKOUT /2 mode	Specifies whether the XCLKOUT divide by 2 mode has been implemented
1	XLOCK	Not implemented. Default of 0 is read.
0	Write buffer	Specifies whether write buffering has been implemented

3.5 Mapping Memory Bus Accesses to the External Interface (XINTF)

The XINTF must appear to the memory bus as standard SARAM blocks, regardless of the word width and protocol of the peripherals attached to them.

The bus size can be dynamically selected by the $\overline{\text{XSIZE32}}$ and $\overline{\text{XSIZE8}}$ signals. The combinations are:

Bus Width	$\overline{\text{XSIZE32}}$	$\overline{\text{XSIZE8}}$
16-bit bus, XD[15:0]	1	1
8-bit bus, XD[7:0]	1	0
Undefined	0	0

When 32-bit bus operation is supported, two write strobes are available; otherwise, only one write strobe is supported:

Bus Width	Write Strobes
16-bit bus, XD[15:0]	$\overline{\text{XWE}} \rightarrow$ XD[15:0]
8-bit bus, XD[7:0]	$\overline{\text{XWE}} \rightarrow$ XD[15:0]

3.6 External DMA Support ($\overline{\text{XHOLD}}$, $\overline{\text{XHOLDA}}$)

The XINTF supports direct memory access (DMA) to its local (off-chip) program and data spaces with the $\overline{\text{XHOLD}}$ signal input and $\overline{\text{XHOLDA}}$ signal output. When $\overline{\text{XHOLD}}$ is asserted (low active) a request to the external interface is generated to hold all outputs from the XINTF in high impedance mode. Upon completion of all outstanding accesses to the external interface, $\overline{\text{XHOLDA}}$ is asserted. $\overline{\text{XHOLDA}}$ signals external devices that the external interface has its outputs in impedance mode and that another device can control access to external memory or peripherals.

The HOLD mode bit in the XINTCNF2 register enables the automatic generation of a $\overline{\text{XHOLDA}}$ signal and granting access of the external bus when a valid $\overline{\text{XHOLD}}$ signal is detected. While in HOLD mode, the CPU can continue to execute code from on-chip memory attached to the memory bus. If an attempt is made to access the external interface while $\overline{\text{XHOLDA}}$ is low, a not-ready condition is generated, halting the processor. Status bits in the XINTCNF2 register indicate the state of the $\overline{\text{XHOLD}}$ and $\overline{\text{XHOLDA}}$ signals.

The HOLD mode bit in XINTCNF2 register bit takes precedence over the $\overline{\text{XHOLD}}$ input signal, enabling customer code to determine whether an $\overline{\text{XHOLD}}$ request is to be honored.

The $\overline{\text{XHOLD}}$ input signal is synchronized at the input to the XINTF before any actions are taken. Synchronization is with respect to SYSCLKOUT.

On reset, the HOLD mode bit is enabled, allowing for boot load of external memory using an $\overline{\text{XHOLD}}$ request. If $\overline{\text{XHOLD}}$ signal is active low during reset, the $\overline{\text{XHOLDA}}$ signal is driven low as in normal operation.

During power up, any undefined values in the $\overline{\text{XHOLD}}$ synchronizing latches are ignored and would eventually be flushed out when the clock stabilizes; therefore, synchronizing latches do not need to be reset.

If an $\overline{\text{XHOLD}}$ active low signal is detected, the $\overline{\text{XHOLDA}}$ signal is driven low only after all pending XINTF cycles are completed. A pending XINTF cycle is any cycle that is currently in the XINTF FIFO queue. Any pending CPU cycles (cycles that are not in the FIFO queue but are active on the core memory bus) are blocked and the CPU is held in a not-ready state if they are targeted for the XINTF.

XA, XD, XLONG, XBYTESEL, $\overline{\text{XDS}}$, $\overline{\text{XRD}}$, $\overline{\text{XWE}}$, XRNW, $\overline{\text{XSTRB}}$, $\overline{\text{XIACK}}$, and $\overline{\text{XVECT}}$ are all in high impedance state in HOLD mode.

All other signals remain in their normal operating states.

3.7 MPNMC Mode

The XMPNMC input signal selects the microprocessor (high) or microcomputer (low) mode of operation This signal is sampled on a system reset ($\overline{\text{SYSRS}}$) and is reflected in the MPNMC bit in the XINTCNF2 register and the MPNMC signal, which comes out of the XINTF block. You can modify the state of the MPNMC bit in the XINTCNF2 register and it is also reflected on the MPNMC signal.

Note: After reset operation, the state of the XMPNMC input signal is ignored.

The MPNMC signal that is generated by the XINTF is connected to Zone 7 and any block of on-chip memory that is enabled in microcomputer mode (such as ROM or, to all blocks of memory on a device that may be used to emulate ROM).

3.8 External Visibility Trace Modes

To enable tracing of events on the XINTF, an instrument such as a logic analyzer uses the following signals to decode the various bus cycle types:

- $\overline{\text{XPCDISC}}$. This signal is used to determine if the current program space fetch ($\overline{\text{XPS}}$ and $\overline{\text{XIAQ}}$ active) is the first instruction to be executed after a program or interrupt. This enables the trace mechanism to determine which instructions are flushed in-between a branch instruction or an interrupt occurring ($\overline{\text{XVECT}}$ signal active) and the current instruction in which $\overline{\text{XPCDISC}}$ is active.
- $\overline{\text{XEMUACC}}$. This signal indicates if the current bus cycle is an emulation access or a normal CPU access, enabling the trace mechanism to ignore debug accesses.
- $\overline{\text{XIAQ}}$. This signal indicates that the current program space bus access ($\overline{\text{XPS}}$ active) is an instruction fetch and not a program space data read or write operation initiated by an XPREAD, XPWRITE, or MAC instruction.
- $\overline{\text{XVECT}}$. This signal indicates that the current program space bus access ($\overline{\text{XPS}}$ active) is a vector fetch. The lowest 6 bits of the address bus indicate which vector is being fetched.
- $\overline{\text{XPS}}$. This signal indicates that the current bus access is from program space. This signal is mutually exclusive to the $\overline{\text{XDS}}$ strobe.
- $\overline{\text{XDS}}$. This signal indicates that the current bus access is from data space. This signal is mutually exclusive to the $\overline{\text{XPS}}$ strobe.
- $\overline{\text{XRD}}$. This signal indicates that the current bus access is a read cycle. This signal is mutually exclusive to the $\overline{\text{XWE}}$ strobe.
- $\overline{\text{XWE}}$. This signal indicates that the current bus access is a write cycle. This signal is mutually exclusive to the $\overline{\text{XRD}}$ strobe.

These signals only give visibility to external bus operations. A PC discontinuity address visibility mode as enabled by the AVIS and XEAVIS bits in the XINTCNF2 register allows you to trace internal program execution.

When the XEAVIS and AVIS mode bits are enabled, the XINTF outputs, on the address bus lines XA[21:0], the PC value at all discontinuity points when program executes from internal memory. The PC value is output on four consecutive SYSCLKOUT cycles and is detected by the following signal combinations:

- $\overline{\text{XDS}}$ and $\overline{\text{XPS}}$ both high for duration of visibility cycle.
- $\overline{\text{XRD}}$ toggles (high for first the two SYSCLKOUT cycles, low for the next two).

- Either the $\overline{\text{XPCDISC}}$ or the $\overline{\text{XVECT}}$ signal goes active, depending on whether the discontinuity is a vector fetch or a branch.
- All other signals go into an inactive state.

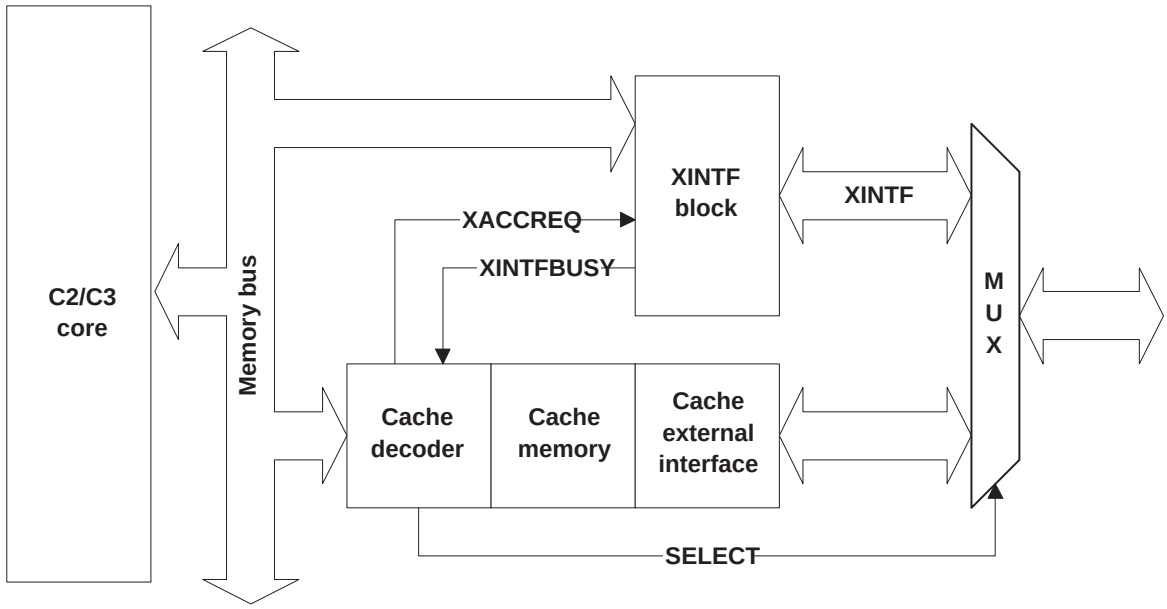
The PC visibility cycle is prioritized like any other XINTF program read cycle but is treated similar to a write bus operation (except no data is driven and the $\overline{\text{XRD}}$ signal is toggled). The PC discontinuity mode can get buffered up to minimize performance impact. If the AVIS buffer is full, the CPU is stalled until the current operation completes and an entry is removed from the AVIS buffer.

An external trace box can then take the PC value and, knowing the source code, can follow the program flow for internal program execution. When the program jumps to external memory execution, the visibility mode is automatically disabled until program execution goes internal again.

3.9 Cache Support

In certain cases, it may be necessary for any other block in the system to gain access to the XINTF external bus pins or disable the XINTF from responding to a memory bus access that is decoded by the XINTF. An example of this is a cache memory block, which must perform a burst access to external memory. To assist in this, two signals are provided, XACCREQ and XINTFBUSY, which can be used to request that the XINTF ignore any active bus activity and complete all pending accesses. See Figure 3–12 for a recommended system diagram.

Figure 3–12. Recommended System Diagram With a Cache Memory Block.



The behavior of the XACCREQ and XINTFBUSY signals is as follows:

1) The cache decoder block detects an access to cache memory space. The decoder generates an XACCREQ signal (active high) which meets the necessary set-up and hold timing within the first bus cycle of the access being placed on the memory bus.

2) The XINTF, on detecting a valid XACCREQ signal, ignores any requested accesses to the XINTF and generates an XINTFBUSY signal (active high). If any accesses are pending (for example, anything in the XINTF FIFO queue), the XINTFBUSY signal is removed when all pending accesses are completed (XINTF FIFO empty).

3) The cache decoder block, on detecting a non-active XINTFBUSY signal, drives the SELECT signal and switches the external bus over to the cache

external interface block. It must do so without creating any bus glitches. On an active XINTFBUSY signal, the XINTF signals are placed in the same mode as an active $\overline{\text{XHOLDA}}$ mode.

4) On completing any cache accesses, the XACCREQ signal is removed at the end of the last bus access to the cache.

5) If a simultaneous XACCREQ and $\overline{\text{XHOLD}}$ request occurs, XACCREQ is given higher priority.

6) The XACCREQ signal is a synchronous input and must meet set-up and hold-times relative to the rising edge of SYSCLKOUT. If the XACCREQ feature is not used, this signal must be tied low within the chip.

3.10 External Interface (XINTF) User-Defined Options

As with the other interface blocks, the XINTF needs some customization for each application. XINTF allows resynthesis of the features needed by your specific application. It does not use a strapping field with the hardware for all the options embedded in each instance, whether they are needed or not. See Table 3–9 for the user-defined options.

Table 3–9. External Interface (XINTF) User-Defined Options

Option	Description
Mapping of XINTF control registers	The mapping of XTIMING0-7, and XINTFCNF0-2 can vary from system to system. See Table 3–1, *External Interface (XINTF) Registers*, on page 3-9 for more information.
IACK instruction	Normally, this instruction behaves like a 16-bit write, except the address value may have no meaning. The write buffer's behavior during an IACK operation is design specific.
Vector fetch	Reset and interrupt vectors can be fetched from external memory via the XINTF. Normally, this operation performs like a 32-bit program read on the bus.
MPNMC emulation	On some external interfaces, you can select either the microprocessor (external boot memory) mode or the microcomputer (on-chip boot memory) mode. The emulation of this mode is system specific.

3.10.1 Removing/Reducing the Write Buffer

The current XINTF code allows for a 3-deep write buffer, with three levels visible to XINTCNF2 write buffer depth bits. The processor stalls when the buffer level equals the buffer mode until all XINTF write accesses are complete, giving the XINTF an extra two-buffer padding. The 3-deep processor-visible write buffering can be disabled through the write buffer depth bits, but the Xfifo VHDL code must be modified if you want to remove the entire 3-deep buffer. The width of the buffer is controlled by a generic in the Xfifo VHDL entity. If you change the width of the buffer, you must change the Xcntrl VHDL code that determines the layout of the buffer (default is the 22-bit address, the 2-bit data, the eight data -select signals, XIACK, and other signals indicating AVIS mode access and whether access came during a DMA/CACHE operation for a total of 71 bits). The FIFO is eight-deep to accommodate the 3-deep buffer; therefore, removing the write buffer implies realigning the size of the FIFO. This changes the equations for the index and read and write pointers in XFIFO RTL.

3.10.2 $\overline{\text{XIACK}}$ Behavior

The IACK instruction with a 16-bit immediate operand behaves like a write operation. The 16-bit immediate data is available on the data bus lines

DWDB[15:0]. The address lines contain the last value on the bus and have no relationship to the IACK instruction. During the IACK operation, the IACK core signal behaves like the data write strobe, DWDS0. These two signals are mutually exclusive. The IACK operation is responded to only by the XINTF and is treated as follows:

- The $\overline{\text{XIACK}}$ strobe signal behaves like the write strobe signal ($\overline{\text{XWE}}$) and has the equivalent timing of a 16-bit write operation in 16-bit bus mode ($\overline{\text{XSIZE32}}$ and $\overline{\text{XSIZE8}}$ are ignored). The address lines on the XINTF contain an undefined value that is the last value on the XA[21:0] bus before the IACK operation.
- An IACK operation to the XINTF stalls the CPU if the write buffer is not empty. It waits for all pending writes to be completed.
- The IACK instruction is not buffered; therefore, the CPU is stalled until the IACK instruction is finished.
- An $\overline{\text{XIACK}}$ operation always uses the strobe timing and the XREADY info from Zone 0.
- Turning off zone 0 DON has no effect on $\overline{\text{XIACK}}$. Turning off the EIACK bit, which is in the XBANK register, disables the $\overline{\text{XIACK}}$ feature.

3.10.3 $\overline{\text{XVECT}}$ Behavior

During a vector-fetch operation, the VECT signal from the T320C2700B0 core is toggled in conjunction with the program read strobes, PRDS0 and PRDS1, and the address bus contains the vector address. For a vector fetch from XINTF space, if any wait states are added, all signals are stretched by the number of wait states. For vector fetches to non-XINTF space (internal memory), the signals go active for four cycles, provided that the AVIS mode has been turned on and the EAVIS bit is enabled. See Table 3–5 on page 3-19 for descriptions of AVIS mode and the EAVIS bit. The vector fetch operation behaves exactly like a program bus read operation on the XINTF, except that fetches from internal memory never activate the $\overline{\text{XPS}}$ signal.

On the XINTF, the vector fetch signal, $\overline{\text{XVECT}}$, behaves as follows:

- $\overline{\text{XVECT}}$ is toggled when the vector fetch is from the XINTF. When AVIS mode is set, $\overline{\text{XVECT}}$ toggles even on vector fetches from internal memory.
- $\overline{\text{XVECT}}$ is an output status signal only; it does not generate any stall or wait conditions.

3.10.4 $\overline{\text{XPCDISC}}$ Behavior

During a branch operation, the $\overline{\text{XPCDISC}}$ signal from the T320C2700B0 core is toggled in conjunction with the program read strobes, PRDS0 and PRDS1,

and the address bus contains the destination address of the branch. For a branch to XINTF space, if any wait states are added, all signals are stretched by the number of wait states. For branches to non-XINTF space (internal memory), the signals go active for four cycles, provided that the AVIS mode has been turned on and the EAVIS bit is enabled. See Table 3–5 on page 3-19 for descriptions of AVIS mode and the EAVIS bit. The branch operation behaves exactly like a program bus read operation on the XINTF, except that fetches from internal memory never activate the $\overline{\text{XPS}}$ signal.

On the XINTF, the branch signal, $\overline{\text{XPCDISC}}$, behaves as follows:

- $\overline{\text{XPCDISC}}$ is toggled when the branch is from the XINTF. When AVIS mode is set, $\overline{\text{XPCDISC}}$ toggles even on branches from internal memory.
- $\overline{\text{XPCDISC}}$ is an output status signal only; it does not generate any stall or wait conditions.

3.10.5 DMA/CACHE

Support for the DMA/CACHE function has been integrated into XINTF in the following ways:

- While a DMA/CACHE request is active, all CPU requests to the XINTF are queued up in the FIFO and are processed once the DMA/CACHE operation is over. This behavior is factored into the XCNTRL block's access arbitration equations for deciding on what CPU requests enter the three latches and the FIFO and what request can be exported immediately.
- To start processing of queued requests once the DMA/CACHE operation is over, the xcnTWL2read (or POP from FIFO) equation in the XCNTRL block looks for the end of a DMA/CACHE operation.
- Since all CPU requests that come during DMA/CACHE requests are queued, the presence of DMA/CACHE operations has to be incorporated into all three OREADY signals in the XCNTRL block.
- To distinguish between CPU requests that came before the DMA/CACHE request from those that came when the DMA/CACHE request was active, each pending access in the Latch/FIFO carries a tag. Setting of the tags (xcnTWG2holdtag and xcnTWG2cachetag) is done in the XCNTRL block. The Tag bits are present in the XLATCH and XFIFO blocks as bits 63 & 64.
- The equations for generating the hold request acknowledge signal (xhlNWG2xholdan) are in the XHOLD block and this block can be done away with if the DMA support feature is not needed.
- The equations for generating the cache request acknowledge signal (xcnTWG2xintfbusy) are in the XCNTRL block.

3.10.6 AVIS Mode

AVIS accesses are treated like program read operations, but have a 3-deep buffer which, like the write buffer, is a part of the FIFO. If the AVIS feature is not needed, modify the following:

- ❑ The XDECODER recognizes AVIS accesses (xdcTWG2pcdisc, xdcTWG2vectf) if the AVIS mode has been enabled.
- ❑ To make AVIS accesses appear as program reads, the AVIS terms are included along with the PRead term in the xcnTBL2decreq signal in the XCNTRL block. This signal goes into the access arbitration process.
- ❑ The book-keeping signals for the AVIS buffer are in the XCNTRL block. These signals are the write-into-buffer signal (xcnTWL2avisbufinp) and the buffer count signal (xcnTBL2aviscount).
- ❑ Bits 69 & 70 of the pending access entries in the XLATCH block and the XFIFO block are the AVIS bits (for pcdisc and vectf respectively).
- ❑ Since the AVIS buffer stalls the CPU once it is full, the xcnTWL2avpulllow and the xcnNWL2avqfull terms are included in the DWOREADY equation in the XCNTRL block.
- ❑ The XAVIS STROBE block is responsible for generating the AVIS mode strobes and can be omitted if the AVIS feature is not needed. Removing the AVIS feature allows reducing the number of latches in the XLATCH block (from 3 to 2) and the number of slots in the FIFO (from 8 to 5). However a thorough analysis regarding the number of latches and slots are required before undertaking such reductions.
- ❑ The equations for the final $\overline{\text{XPCDISC}}$ and $\overline{\text{XVECT}}$ strobes are in the XRWBLK.
- ❑ The AVIS and the EAVIS bits in the XPERREG block can be removed, if the AVIS feature is not needed.

3.10.7 XREADY feature

The active phase of a read or write strobe can be extended by the XREADY signal if the USEREADY bit for a zone is activated. This feature is incorporated into the XINTF at the following locations:

- The XWGEN block has logic for generating a synchronous XREADY signal (xwgTWL2syncxready) from an asynchronous input in the XWGEN block. This logic can be removed if the ready option is not needed.
- The signal that detects the last active cycle (xwgTWL2xdsamp) currently depends on the ready setting. This can be simplified if the ready option is not used. Doing this helps XINTF timing.

3.11 Example External Interface (XINTF) Setup

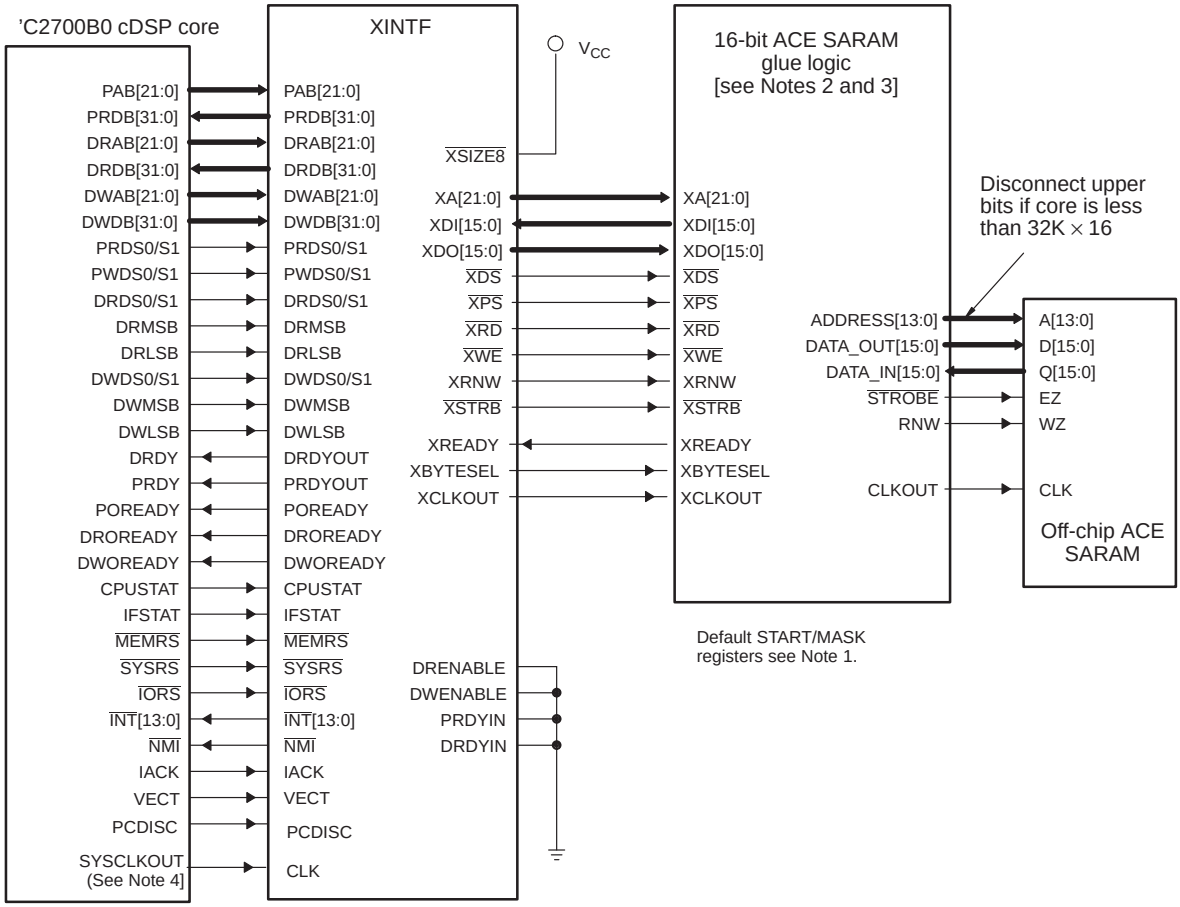
Example 3–1 code shows a configuration for an XINTF and one external off-chip ACE SARAM. Minimal glue logic is needed to interface the SARAM to the XINTF. In the case of an ACE SARAM, the XINTF signals XRNW and $\overline{\text{XSTRB}}$ are connected to WZ and EZ, respectively, on the SARAM core to enable it for reads and writes. You need to use the $\overline{\text{XRD}}$ and $\overline{\text{XWE}}$ signals if your peripheral has separate pins for read enable and write enable.

A peripheral's timing characteristics determine the configuration of the XTIMING registers (see section 3.4.1, *XINTF Timing Registers (XTIMING)*, on page 3-15). An ideal peripheral needs no wait states; however, the XINTF timing cannot be set for 0 wait states. A typical ACE SARAM needs one cycle of setup time for reads and one cycle of hold time for writes. XREADY can be disabled because the setup and hold times defined in the XTIMING register are sufficient for the memories to function properly. XREADY is usually implemented for slower peripherals at the cost of additional glue logic. The resulting XTIMING register configuration for the ACE SARAM is 0001 0000 0000 0001. See Example 3–2 for an example 16-bit memory glue logic VHDL architecture.

Example 3–1. Example VHDL Glue Logic Between External Interface (XINTF) and 16-Bit Off-Chip ACE Memory

```
architecture rtl of xmem_glue_logic is
begin

   CLKOUT <= XCLKIN;
   ADDRESS <= XA(13 downto 0);
   DATA_OUT <= XDO;

   process (XRNW, XSTRBn, XDSn, XPSn, XMEM_DON,
XMEM_PON,
            XMEM_DSTRT, XMEM_PSTRT, XA)
   begin
      if (((XDSn = '0') and (XMEM_DON = '1') and
             (XA(21 downto 10) = XMEM_DSTRT)) or
             ((XPSn = '0') and (XMEM_PON = '1') and
             (XA(21 downto 10) = XMEM_PSTRT))) then
         RNW <= XRNW;
         STROBEn <= XSTRBn;
      else
         RNW <= '1';
         STROBEn <= '1';
      end if;
   end process;

         -- SARAM has 16-bit bus
   XDI <= DATA_IN;

end rtl;
```

Example 3–2. Example of 16-Bit Memory Glue Logic VHDL Architecture



Notes: 1) tieTBG2DEFXZ[7:0]strt: in std_logic_vector (11:0)
tieTBG2DEFXZ[7:0]mask: in std_logic_vector (9:0)

The lower 2 bits of the 12-bit start registers are the DON and PON bits for that zone. The upper 10 bits specify the 4K address boundary to which the zone is mapped. The 10-bit range registers can be used to specify how many address bits of the start address are actually used in the zone decode. This facility allows you to increase the zone size above the default 4K value. See Section 3.3.1 for more information on configuring the registers.

2) See Example 3–1 for a possible implementation of VHDL glue logic architecture
3) All interface signals to an off-chip memory must be brought out as pins on the cDSP device.
4) SYSCLKOUT is not connected directly to CLK on the XINTF, SYSCLKOUT must go through a clock tree synthesis (CTS) buffer.

3.12 External Interface (XINTF) Timing

Figure 3–13 through Figure 3–20 show the timing for XINTF operations.

Figure 3–13. XINTF Read Operation (Setup = 1, Active = 0, Hold = 1) Timing

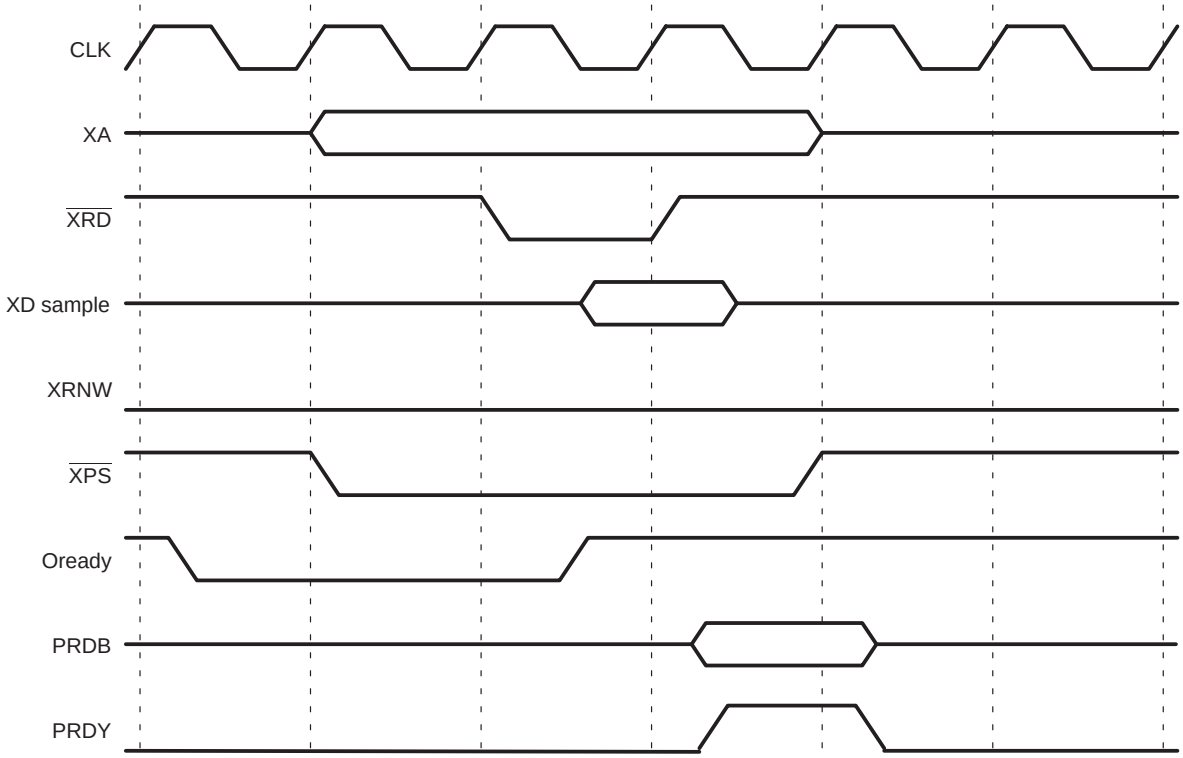


Figure 3–14. XINTF Read Operation (Setup = 1, Active = 0, Hold = 0 with Xready) Waveform

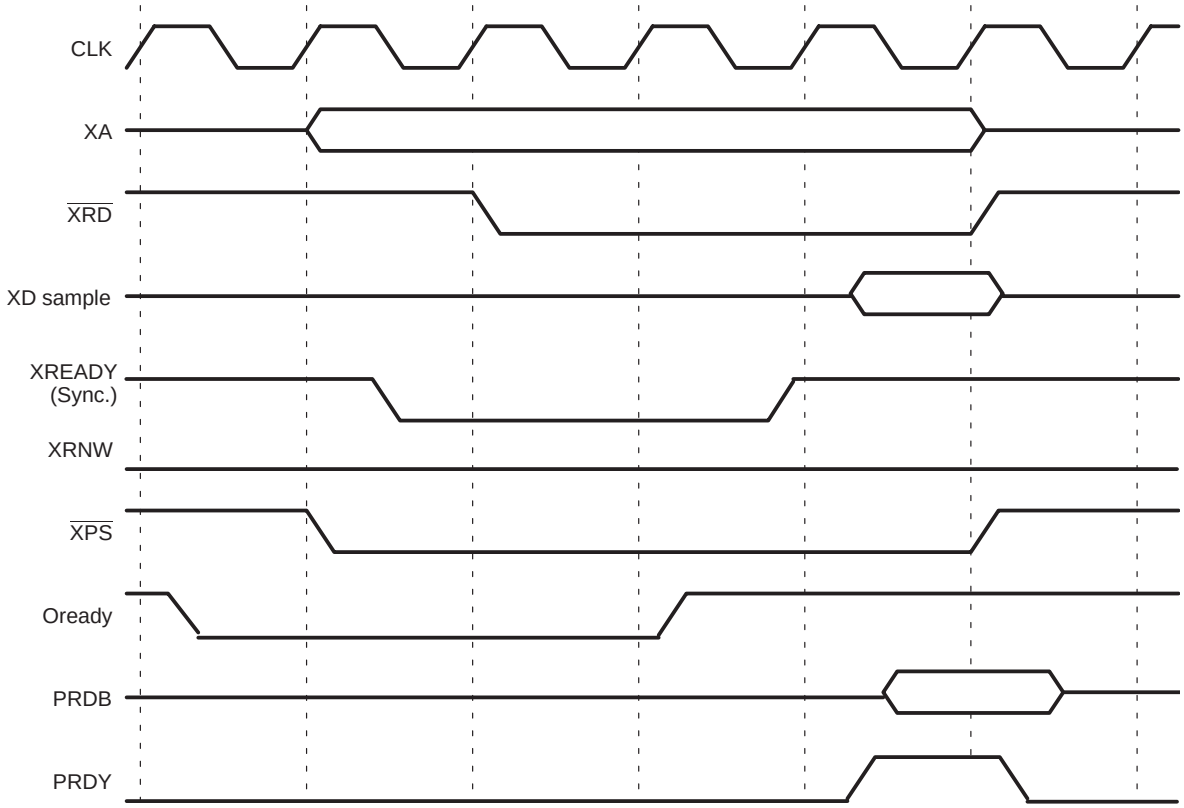


Figure 3–15. XINTF Write Operation (Setup = 1, Active = 0, Hold = 0, Mode = 0) Waveform

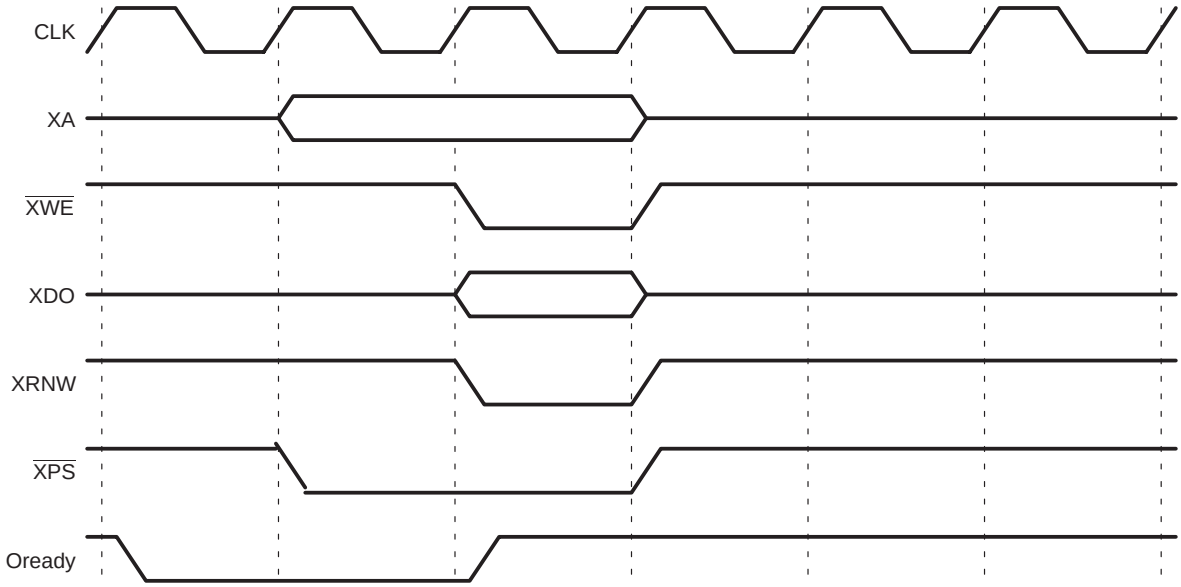


Figure 3–16. External Interface Generic Read Cycle Waveform

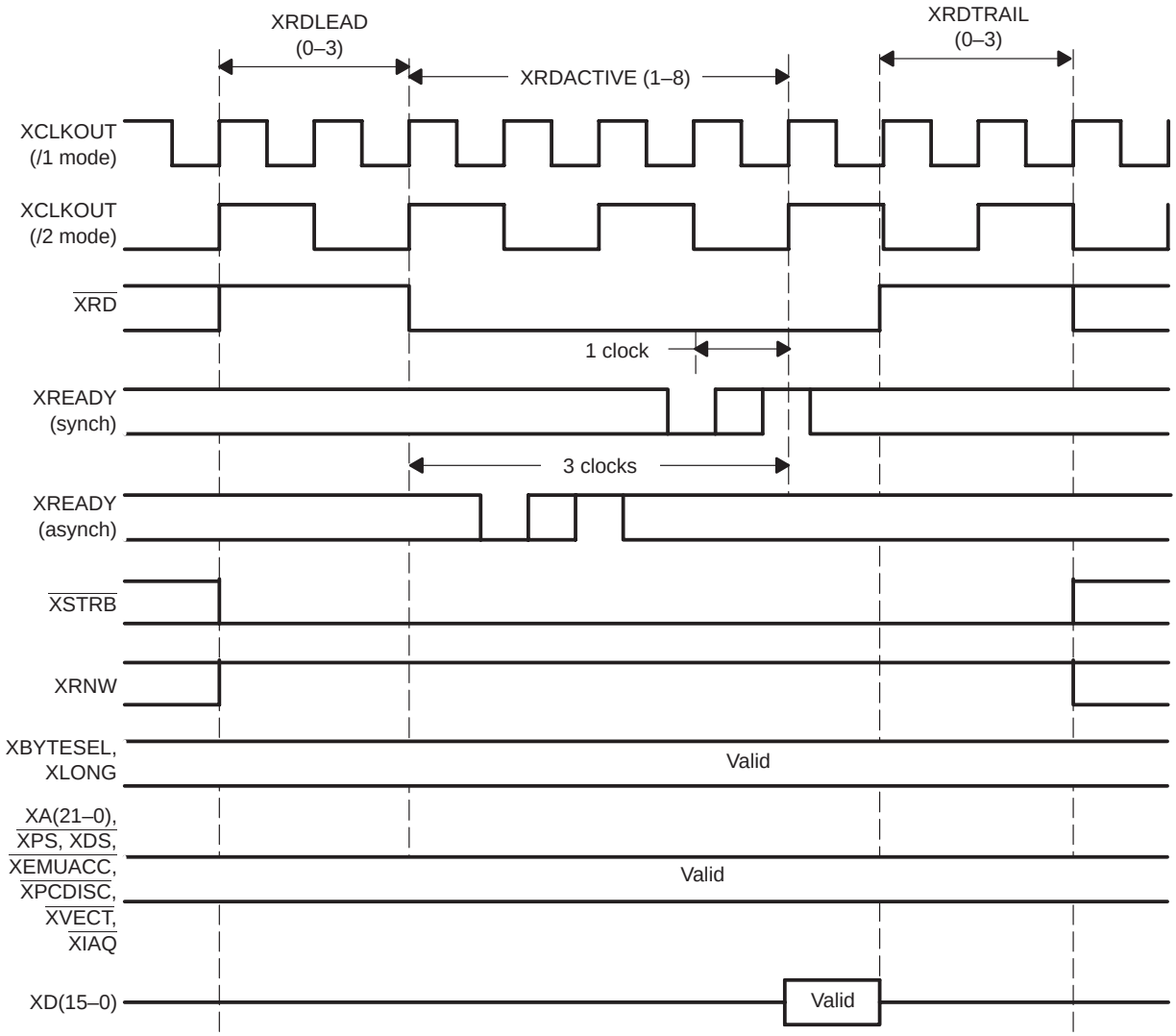


Figure 3–17. External Interface Generic Write Cycle Waveform

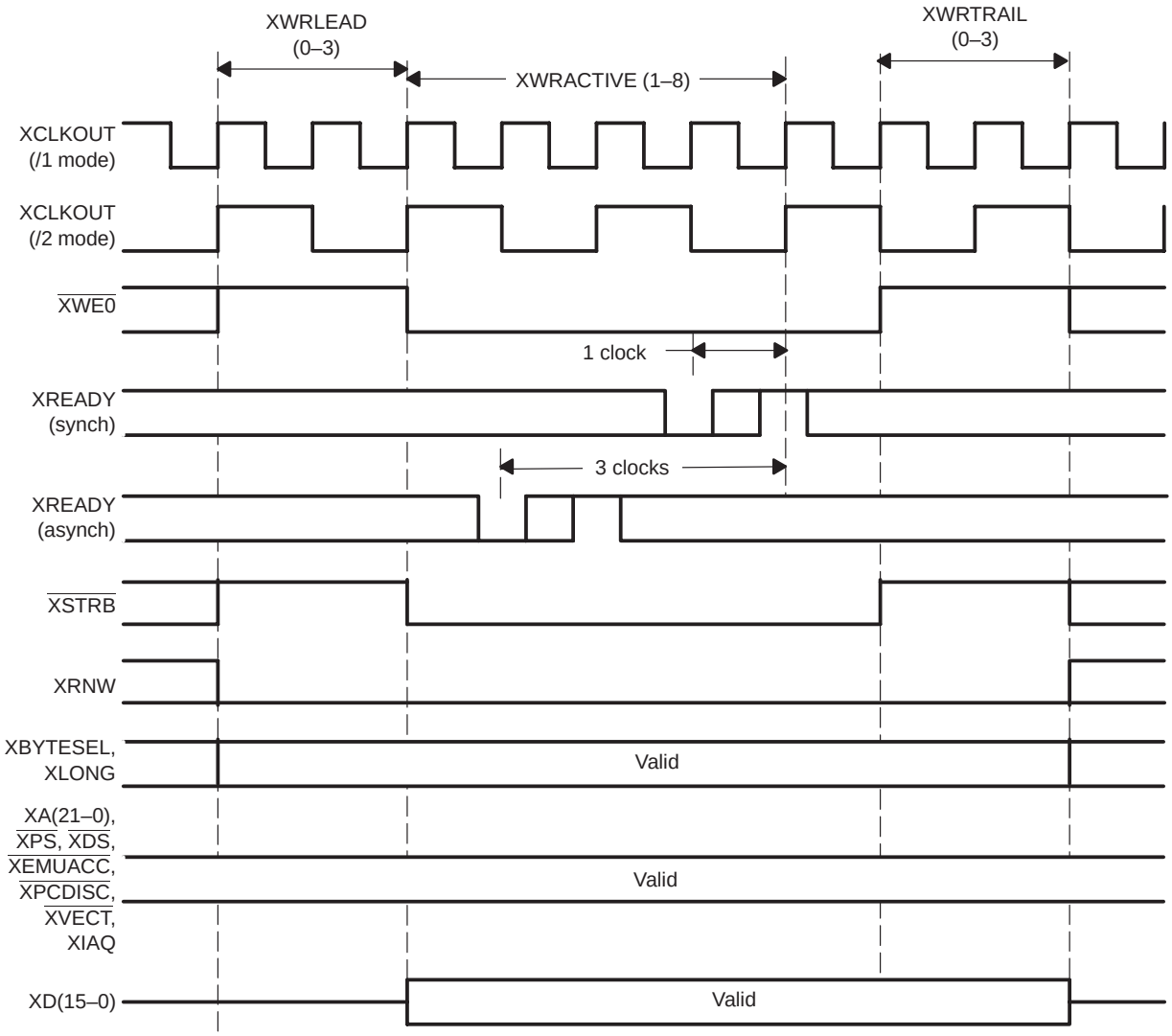


Figure 3–18. External Interface Generic IACK Waveform

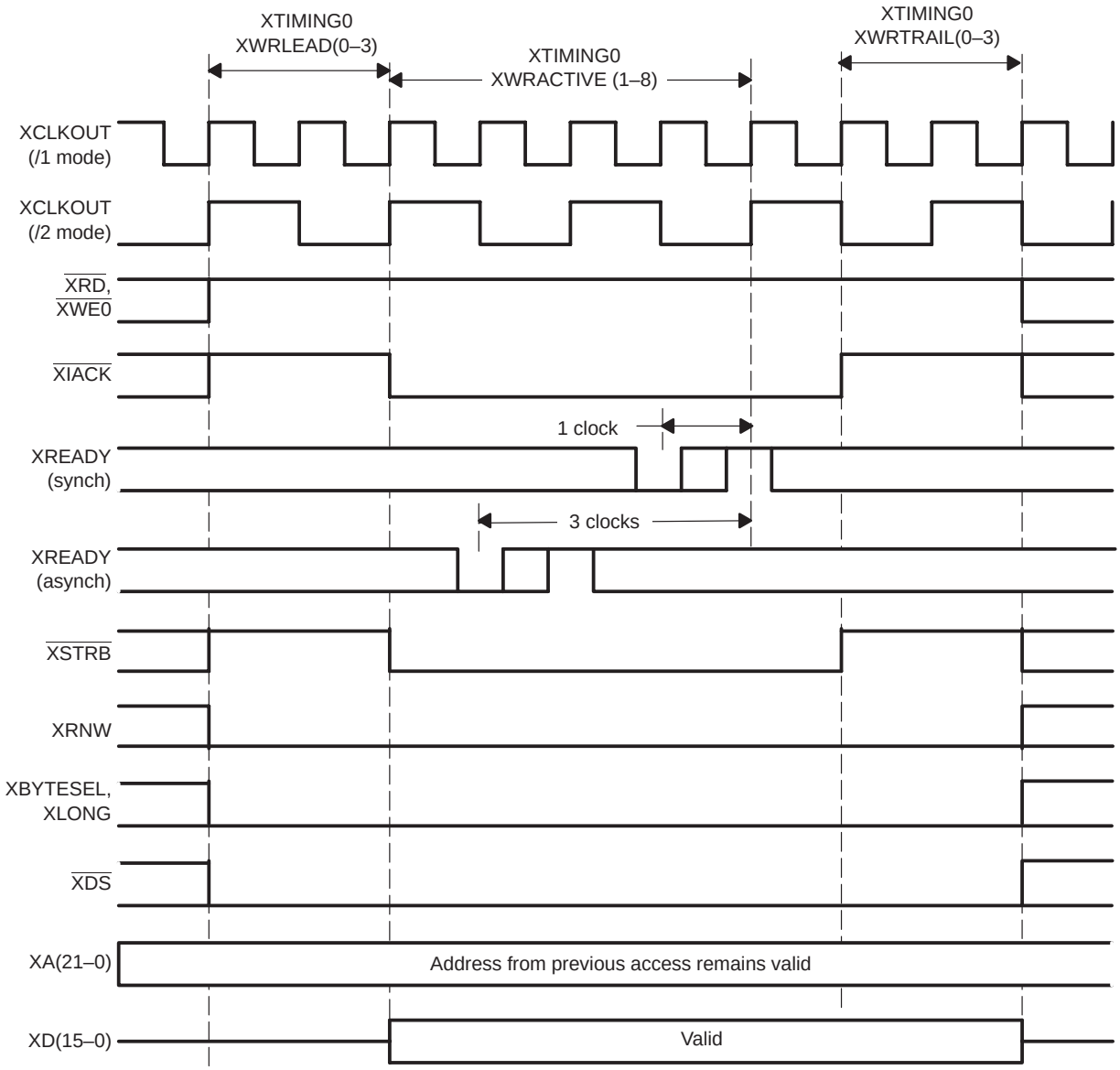


Figure 3–19. External Interface Generic Hold Waveform

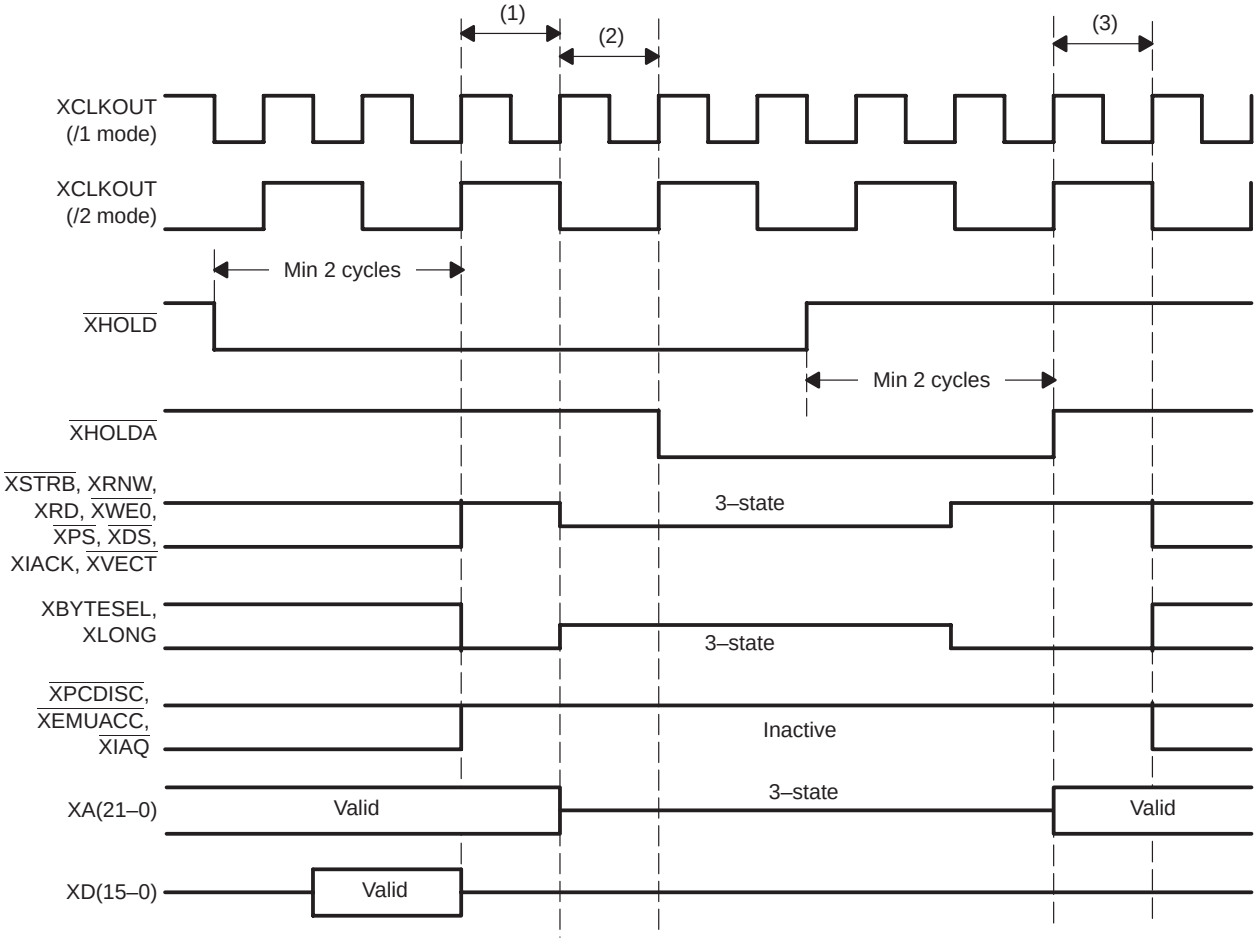
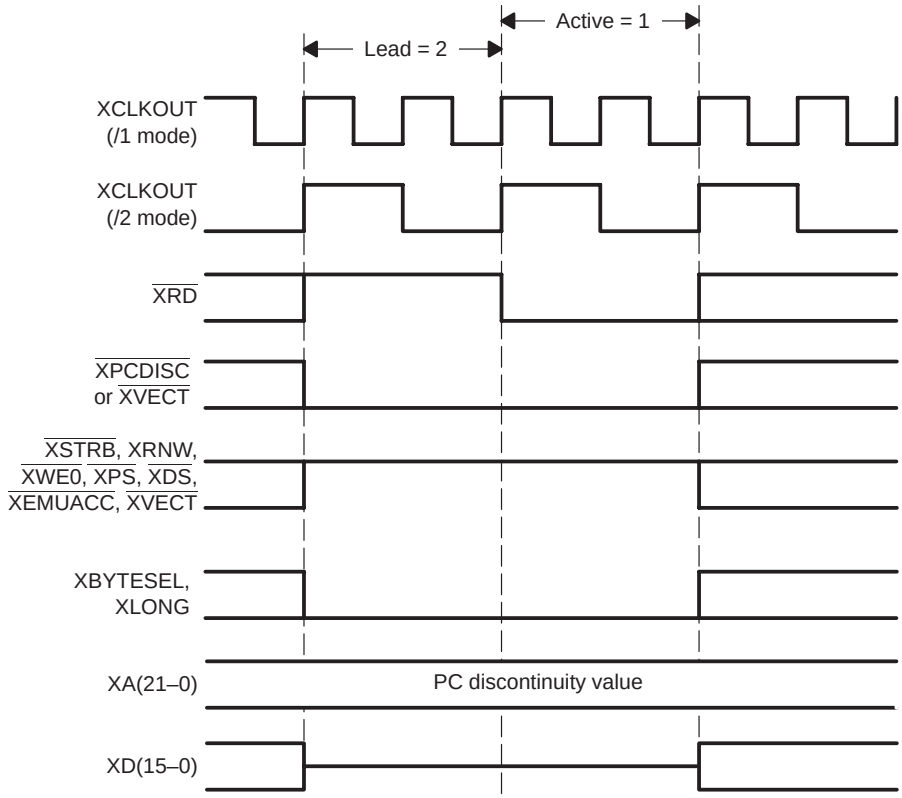


Figure 3–20. External Interface Generic Visibility Mode Waveform



Chapter 4

Timer

The timer is a programmable peripheral that is used to generate pulses or to time events. It can be used to generate interrupts after a defined number of clock cycles. It is an optional feature, separate from the ’C2700B0 core and the external interface.

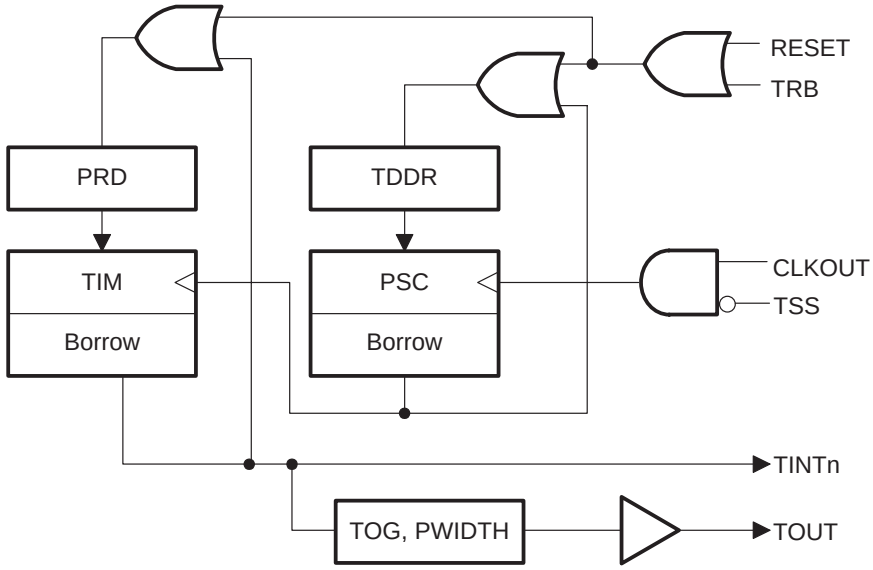
Topic **Page**

4.1 Timer Operation

The timer (Figure 4–1) has an 8-bit prescaler that is used to divide down the period of the clock into the clock module and change the rate at which the timer is incremented. It can be clocked at a rate between 1 and 1/256 of the timer clock input rate, depending upon the timer prescaler divide-down value. The timer is clocked by CLKOUT from the CPU. The timer has a down counter that can be stopped, restarted, reset, or disabled by specific status bits.

You can use the timer to generate periodic CPU interrupts. You can use multiple timers, each with its own interrupt pin. When the timer decrements to 0, it sets the timer interrupt flag (TIF) in the timer control register (TCR). If the timer interrupt enable (TIE) bit is also set, the timer interrupt (TINTn) signal is asserted. This interrupt signal can then be connected to any one of the core interrupt input lines. The timer can have an optional device pin associated with it (TOUT) that can be asserted high or low for between one and eight CLKOUT cycles when the timer reaches 0, or can simply be toggled. Control register bits define the operation of the timer out (TOUT) pin when the counter reaches 0.

Figure 4–1. Timer Block Diagram



The $\overline{\text{TINT}}$ rate is given by:

Equation 4–1. Timer Interrupt Rate

$$\text{TINT rate} = \frac{1}{t_{c(CO)} \times u \times v} = \frac{1}{t_{c(CO)} \times (\text{TDDR} + 1) \times (\text{PRD} + 1)}$$

where:

$t_{c(CO)}$ = period of CLKOUT

u = timer divide-down register (TDDR) contents + 1

v = timer period register (PRD) contents + 1

In Equation 4–1, the timer interrupt rate equals the timer clock input source frequency ($1/t_{c(CO)}$) divided by two independent factors (*u* and *v*). Each of the two divisors is implemented with a down counter and a period register. The counter and period registers for *u* are the prescaler counter (PSC) and TDDR, respectively, both 8-bit fields of the TPR. The counter and period registers for *v* are the timer counter register (TIM) and PRD, respectively. Both are16-bit registers mapped to I/O space.

Each time a counter decrements to 0, a borrow is generated on the next CLKOUT cycle and the counter is reloaded with the contents of its corresponding period register. The contents of the PRD are loaded into the TIM when the TIM decrements to 0 or when a 1 is written to the timer reload bit (TRB). Similarly, the PSC is loaded with the value in the TDDR when the PSC decrements to 0 or when a 1 is written to the TRB.

In the cycle when the TIM decrements to 0, it generates a pulse that has a duration equal to two CLKOUT periods ($t_{c(CO)}$). This is the raw output of the timer, which is sent to the CPU as a $\overline{\text{TINT}}$ signal. This raw timer output is also used to generate the signal at the TOUT pin, the final output depends on the value of the TOG, POL, and PWIDTH bits in the TCR.

Here is a typical series of events for the timer:

1) PSC decrements on each succeeding CLKOUT pulse until it reaches 0.

2) On the next CLKOUT cycle, the TDDR loads the new divide-down count into the PSC and the TIM decrements by 1.

3) The PSC and the TIM continue to decrement in the same way until the TIM decrements to 0.

4) In the cycle when the timer reaches 0, a $\overline{\text{TINT}}$, which is a low-going pulse that is two CLKOUT cycles long, is sent to the CPU and to the TOUT signal generation logic.

5) In the next cycle after the timer reaches 0, the new timer count is loaded from the PRD into the TIM and the PSC is decremented by 1.

4.2 Remapping the Timer Registers

On the ’C2700B0 core it is possible to remap the timer registers anywhere in the data memory space on any 64 × 16 word boundary. It is also possible to disconnect the timer registers from the memory map altogether. The mapping of these registers is controlled by the memory-map configuration registers listed in Table 4–1. These registers are mapped in the reserved emulation register space.

Table 4–1. Timer Memory-Map Configuration Registers

Name	Description	Address Offset	Figure
DSTRT_TIMER	Timer block data space starting address	00h	Figure 4–2
MCTL_TIMER	Timer block memory control register	02h	Figure 4–3

The DSTRT_TIMER (Figure 4–2) maps the beginning of the timer register block in data space. The higher bits of the data read address bus, DRAB[21:6], and data write address bus, DWAB[21:6], are compared with the DSTRT_TIMER[15:0] bits to determine whether the interface is selected.

In the MCTL_TIMER (Figure 4–3), when data external on (DON) bit 1 is set, the timer register block is mapped to data space. The value of DSTRT_TIMER can be modified when this bit is cleared to 0.

Figure 4–2. Timer Block Data Space Starting Address Register (DSTRT_TIMER)

15–0
DSTRT_TIMER
R/W

Note: R = read access; W = write access

Figure 4–3. Timer Block Memory Control Register (MCTL_TIMER)

15–2	1	0
Reserved	DON	Res
X	R/W	X

Note: R = read access; W = write access; X = don’t care

4.3 Timer Registers

The timer is controlled by the registers in Table 4–2.

Table 4–2. Timer Registers

Name	Description	Address Offset
PRD	Timer period register	02h
TCR	Timer control register	04h
TIM	Timer counter register	00h
TPR	Timer prescale register	06h

4.3.1 Timer Period Register (PRD)

The 16-bit PRD holds the next starting count for the timer; it supplies the TIM with the next value to decrement. When the TIM decrements to 0, a TINTn signal is generated. At the start of the next timer input clock (the output of the timer prescaler) cycle, the contents of PRD are loaded into TIM. The PRD contents are also loaded into TIM when you set the TRB in the TCR.

With the timer divide-down value at 0, you can generate a timer interrupt request every (PRD + 1) CLKOUT cycles. Because you can program the PRD from 0 to 65 535 (FFFFh), you can generate the interrupt every 1 to 65 536 cycles.

With a nonzero divide-down, you can further decrease the timer interrupt rate. The number of CLKOUT cycles between interrupts is then (PRD + 1) × (TDDR + 1), where TDDR is the divide-down value determined by bits 0–7 of the timer prescale register (TPR).

At reset, the PRD is set to hold its maximum value of FFFFh.

You control the duration of the timer's current period and its next periods. You can write to or read from the TIM and PRD on any cycle. You can monitor the count by reading from the TIM and writing the next counter period to the PRD without disturbing the current timer count. The timer starts on the next period after the current count is complete. If both the PRD and TIM are loaded with new values, the timer begins to decrement using the new period without generating an interrupt.

At reset, the TIM and PRD are both set to FFFFh; the TIM begins to decrement only after reset is deasserted.

If you are not using the timer, you can mask the timer interrupt request using the TIE bit and use the PRD as a general-purpose data memory location. If you

use the timer interrupt request, you must program the PRD and TIM before unmasking it to avoid unwanted interrupts.

The address offset of the PRD is +02h.

4.3.2 Timer Control Register (TCR)

The 16-bit TCR contains the control bits that:

- Define the divide-down value of the timer
- Start and stop the timer
- Reload the timer
- Specify the current count in the prescaler counter
- Control the mode of the timer
- Interrupt flag and interrupt enable
- Control the action of the timer output pin when the counter reaches 0

The address offset of the TCR is +04h. The TCR is shown in Figure 4–4 and described in Table 4–3.

Figure 4–4. TCR

15	14	13–12	11	10	9	6	5	4	3	2	1	0
TIF	TIE	Res	FREE	SOFT	PWIDTH	FRCEN	TRB	TSS	TOG	POL	FORCE	OUTSTS
R	R/W	R	R/W	R/W	R/W	R/W	W	R/W	R/W	R/W	W	R

Note: R = read access; W = write access

Table 4–3. TCR Bit Descriptions

Bit	Name	Function
15	TIF	Timer interrupt flag. This flag is set to 1 when the timer decrements to 0. This flag can be cleared in software by writing a 1 to it, but this flag can only be set when the timer decrements to 0. Writing a 1 to this bit clears it, writing a 0 has no effect.
14	TIE	Timer interrupt enable. If the timer decrements to 0 and this bit is set to 1, the timer asserts the $\overline{\text{TINT}}$ signal.
13–12	Res	Reserved
11	FREE	This bit is used in conjunction with the SOFT bit to determine the state of the timer when a halt is encountered in the high-level language debugger. When the FREE bit is cleared, the SOFT bit selects the emulation mode. See Table 4–4 on page 4-9. FREE = 0 The SOFT bit selects the timer mode. FREE = 1 The timer runs free regardless of the SOFT bit.

Table 4–3. TCR Bit Descriptions (Continued)

Bit	Name	Function
10	SOFT	This bit is used in conjunction with the FREE bit to determine the state of the timer when a halt is encountered in the high-level language debugger. When the FREE bit is cleared, the SOFT bit selects the emulation mode. Note that in SOFT mode, the timer generates a TINTn before shutting down (since reaching 0 is the interrupt causing condition). See Table 4–4 on page 4-9. SOFT = 0 The timer stops the next decrement of the TIM. SOFT = 1 The timer stops when the TIM decrements to 0.
9–7	PWIDTH	Output pulse width. This field is used to set the pulse width of the TOUT when not in toggle mode. The output pulse width is (PWIDTH + 1) CLKOUT cycles.
6	FRCEN	Force enable. This bit is used in conjunction with the FORCE bit. When a 1 is simultaneously written to the FRCEN bit, the FORCE bit forces the TOUT to the value written to the FORCE bit.
5	TRB	Timer reload bit. This bit resets the timer. When this bit is set to 1, the TIM is loaded with the value in the PRD and the PSC is loaded with the value in the TDDR. This bit is always read as 0.
4	TSS	Timer stop status bit. This bit stops or starts the timer. At reset, this bit is cleared to 0 and the timer immediately starts timing. TSS = 0 The timer is started. TSS = 1 The timer is stopped.
3	TOG	Timer output toggle mode enable. This bit sets the timer in toggle mode. TOG = 0 The TOUT is not in toggle mode and the POL bit is used to determine the action when the counter reaches 0. TOG = 1 The TOUT is in toggle mode and the output switches every time the counter reaches 0.
2	POL	Timer output polarity. This bit determines the pulse value on the TOUT pin. POL = 0 The TOUT is normally high and pulses low for (PWIDTH + 1) CLKOUT cycles. POL = 1 The TOUT is normally low and pulses high for (PWIDTH + 1) CLKOUT cycles.

Table 4–3. TCR Bit Descriptions (Continued)

Bit	Name	Function
1	FORCE	Force output. This bit forces the TOUT to the value written to the bit when a 1 is also simultaneously written to the FRCEN. If the output pin is forced by writing to the FORCE bit in the same cycle that the timer counter is also trying to change the timer output, the FORCE bit value has priority. The FORCE bit is always read as 0.
0	OUTSTS	Output status. This bit reflects the current status of the TOUT.

Table 4–4. Timer Emulation Modes

FREE bit	SOFT bit	Timer Emulation Mode
0	0	Stop after the next decrement of the TIM (hard stop)
0	1	Stop after the TIM decrements to 0 (soft stop)
1	0	Free run
1	1	Free run

4.3.3 Timer Counter Register (TIM)

The 16-bit TIM holds the current count of the timer. The TIM decrements by 1 every (TDDR + 1) clock cycles, where TDDR is the timer prescaler divide-down value determined by bits 0–3 of the TCR. When the TIM decrements to 0, the TINTn bit of the interrupt flag register (IFR) is set (a timer interrupt is pending) and a pulse is sent to the TOUT pin.

You can write values from 1 to 65 535 (FFFFh) to the TIM. At reset, the TIM is set to hold its maximum value of FFFFh. The TIM can be read from as well as written to.

The address offset of the TIM is +0h.

4.3.4 Timer Prescale Register (TPR)

The 16-bit TPR contains the control bits that:

- ❑ Define the divide-down value of the timer
- ❑ Specify the current count in the prescaler counter

The address offset of the TPR is +6h. The TPR is shown in Figure 4–5 and described in Table 4–5.

Figure 4–5. TPR

15–8	7–0
PSC	TDDR
R	R/W

Note: R = read access; W = write access

Table 4–5. TPR Field Descriptions

Bits	Name	Function
15–8	PSC	Timer prescaler counter. These bits specify the count for the timer. When the PSC is decremented past 0 or the timer is reset, the PSC is loaded with the contents of the TDDR and the TIM is decremented by 1. The PSC can be checked by reading the TCR, but it cannot be set directly. The PSC must get its value from the TDDR. At reset, the PSC is set to 0.
7–0	TDDR	Timer divide-down register. These bits specify the timer divide-down ratio (period) for the timer. When the PSC is decremented past 0 or the timer is reset, the PSC is loaded with the contents of the TDDR.

4.4 Timer at Hardware Reset

On a device reset, the CPU sends a RESET signal to the peripheral circuits, including the timer. The RESET signal has the following consequences on the timer:

- The registers TIM and PRD are loaded with their maximum values (FFFFh).
- All the bits of the TCR and TPR are cleared to 0 with the following results:
  - The divide-down value is 0 (TDDR = 0 and PSC = 0).
  - The timer is started (TSS = 0).

Chapter 5

T320C2700B0 Test Requirements and Considerations

To access and test the ’C2700B0 core, the ’C2700B0 cDSP device requires certain configurations and connections. This chapter describes the minimum requirements for testing your cDSP device. It also describes factors that are not requirements but which you may want to consider in designing your device.

Topic **Page**

5.1 cDSP Testing Overview

In any ’C2700B0 cDSP device, TI is responsible for testing the ’C2700B0 core and all TI-defined memory interface components. You are responsible for providing TI with tests only if any of your components must be tested.

Different tests run on different test modes. The ’C2700B0 core test modes are:

- **Core functional test mode (COREFTEST).** In this test mode, the core is completely isolated from the logic connected to it. B0 and B1 are the only memories that the core can access in this mode. All interrupts, reset, NMI, and READY signals from other memories are isolated from the core.
- **Memory interface functional test mode (MEMXFTEST).** In this test mode, all the components connected to the memory interface can be accessed by the core. Interrupts, NMI reset, and others are still isolated from the core.
- **Core automatic test pattern generation (COREATPG)**. This mode is defined for ATPG of the core.
- **Peripheral automatic test pattern generation (PERIATPG)**. This mode is defined for ATPG of components connected to the core.
- **Slave mode**. In this test mode, the core is completely inactive and does not respond to any of the normal input signals. The outputs are in an unknown state. Only the test mode status signals contain valid output values, as shown in Table 5–1.
- **Normal**. This is any mode other than the test modes defined above.

Table 5–1 shows the output values for the ’C2700B0 test modes.

Table 5–1. Test Modes of Operation

Operating (Test) Mode	XLOGOFF Output	COREATPG Output	PERIATPG Output	MEMXFTEST Output	COREFTEST Output
COREFTEST	0	0	0	0	1
MEMXFTEST	0	0	0	1	0
COREATPG	1	1	0	0	0
PERIATPG	0	0	1	0	0
Slave mode	1	0	0	0	0
Normal	0	0	0	0	0

All ouput signals are valid in the COREFTEST and MEMXFTEST test modes. These signals, except the test mode status signals, are invalid in the COREATPG and PERIATPG test modes. In the COREATPG mode, the outputs from the core are invalid, therefore, all memory interface components and XINTF must be completely inactive in this mode. This is simple if SYSCLKOUT is used for clocking the peripheral logic, since SYSCLKOUT is turned off in this mode. In addition, memory interface components must use XLOGOFF to asynchronously place the output drivers in high impedance mode.

5.2 T320C2700B0 Test Requirements

This section describes the minimum requirements for testing your device. If it does not meet these requirements, TI cannot run the necessary tests on your device.

5.2.1 B0 and B1 SARAM

The ’C2700B0 core must have at least 256 ×16 words in each of the two SARAM blocks called B0 and B1. These blocks are necessary for TI to run different tests on the ’C2700B0 core. Memory block B0 is mapped to both program and data space in the ’C2700B0 CPU. Memory block B1 is mapped only to data space.

The ’C2700B0 core can be tested with block sizes that are greater than 256 × 16 words of B0 and 256 × 16 words of B1. However, with larger sizes, only specific combinations of B0 and B1 can be emulated on the system emulation device, ’C2700B0-E1. These specific combinations of B0 and B1 are:

- B0 size of 256, 512, or 1K words

  B0 is mapped to program space starting at address 0 and to data space starting at address 400h.

- B1 size of 256, 512, or 1K words

  B1 is mapped to data space at various addresses. B1 of size 256 decodes at multiple addresses (mirroring). These data space addresses are 0h–FFh, 100h–1FFh, 200h–2FFh, and 300h–3FFh. B1 of size 512 decodes at addresses 0h–1FFh and 200h–3FFh. B1 of size 1K decodes at address 0h–3FFh.

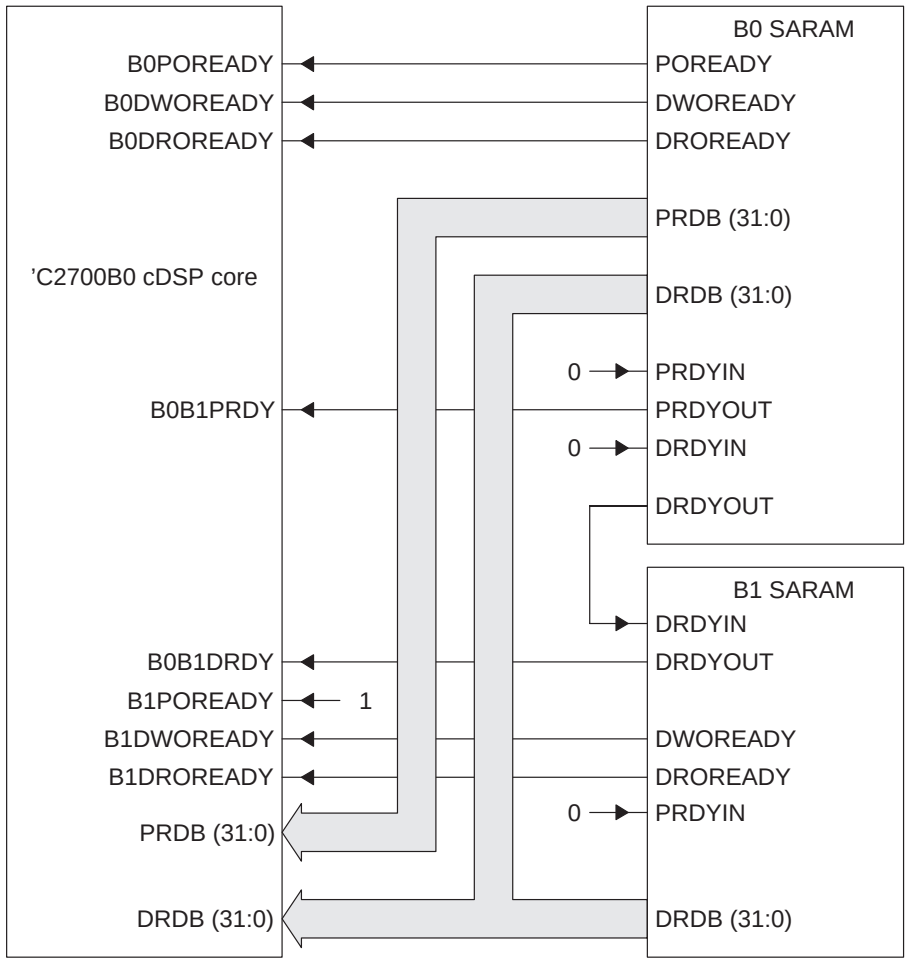
To enable TI to run tests on the ’C2700B0 core, you must connect the B0 and B1 memory blocks to the B0/B1-specific OREADY, PRDY, and DRDY signals as shown in Figure 2–13. These B0 and B1 connections allow the core to disassociate from the surrounding logic, thereby allowing access to B0 and B1 without interference from the user logic.

For descriptions of the OREADY, PRDY, and DRDY core input and memory wrapper signals, as well as other memory signals, see section A.1.1, *Memory Interface Signals*.

Note:

OREADY refers to all memory interface ready signals. These include the DROREADY(13:0), DWOREADY(13:0), and POREADY(13:0) signals, as well as those used by the B0 and B1 memory blocks.

Figure 5–1. B0 and B1 SARAM Connections to the TMS320C2700B0 Core



Note: Not all connections are shown. See Figure 2–7 on page 2-15 for a complete connection diagram.

In Figure 5–1, the B0/B1 connection allows the core to dissociate itself from the surrounding logic, thereby, gaining access to B0 and B1 without interference from the user logic.

Note:

READY refers to all memory interface READY signals. These include the DROREADY [13:0], DWOREADY [13:0], and POREADY [13:0] signals, as well as those used by the B0 and B1 memory blocks.

5.2.2 Connection of Signals

In order for TI to run tests on the ’C2700B0 cDSP device, the JTAG port signals, the core input signal, and the output of the clock tree synthesis (CTS) buffer of SYSCLKOUT must be brought out directly from the ’C2700B0 core to the device pins. In other words, each signal must be connected to its corresponding JTAG port pin. Signals that must be brought out directly from the core to the device pins are:

- $\overline{\text{TRST}}$
- TCK
- TMS
- TDI
- TDO
- ET0 (also known as EMU0)
- ET1 (also known as EMU1)
- SYSCLKOUT

See Appendix A, *Signal Descriptions (TSC6000 ASIC Library)*, for descriptions of these signals.

ET0 is a bidirectional signal using ET0I, ET0O and ET0OEN. ET1 is a bidirectional signal using ET1I, ET1O and ET1OEN.

The core input signal, SLAVEIN, can either be brought out directly from the core to the device pins or tied low. To test the core and connect to the XDS510™ emulator, you must satisfy one of these two conditions for SLAVEIN.

Any logic placed between the core and the JTAG pins can make the core untestable and disable access to the emulator. Therefore, the JTAG signals must be assigned to dedicated pins. Any other scheme must be a joint development with TI.

5.3 T320C2700B0 Considerations

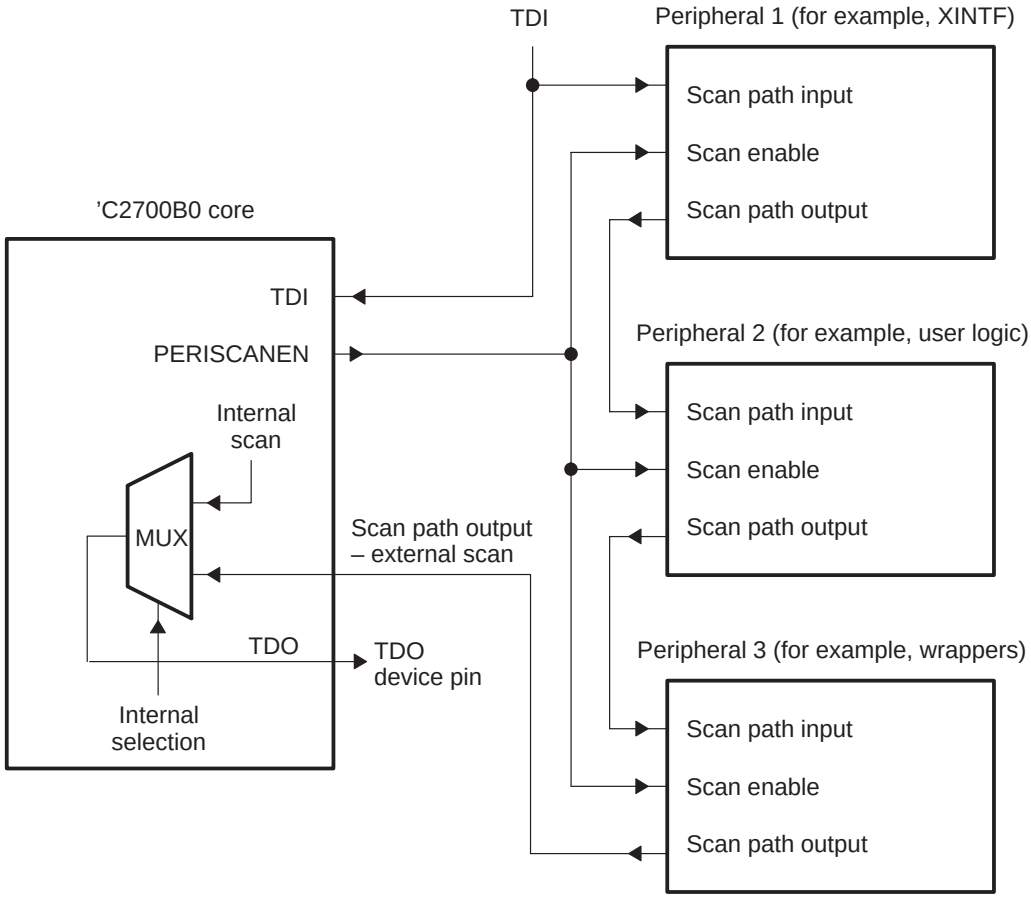
To aid you in designing your ’C2700B0 cDSP device, the following sections describe various test considerations, as well as design considerations for various signal output states.

5.3.1 Peripheral ATPG (PERIATPG) Considerations

PERIATPG is a test method used for testing the memory interface components. To avoid confusion, the memory interface components here are referred to as peripherals. PERIATPG is used when you want the peripherals tested for a high fault grade. Peripheral ATPG is also referred to as scan test of the memory interface components.

Setting up a PERIATPG scan chain is recommended by TI. For TI to run a scan test on the peripherals, you must first connect them in a specific order. Currently, the ’C2700B0 cDSP core supports scan testing of only one external scan chain. Therefore, all components being tested (TI components, as well as your own logic) must be connected to the memory interface in a single scan chain to run a scan test on the peripherals (see Figure 5–2).

Figure 5–2. Connection of Peripherals During PERIATPG



To perform peripheral ATPG, TI provides you with a test description language (TDL), which is a test pattern format accepted by TI's internal set of tools. This particular TDL initializes the core and sets it up so that the PERIATPG can be performed on the peripherals.

The main TDL operations during a scan test of the memory interface components are as follows:

1) The peripheral ATPG mode is set up through the JTAG port.

2) The peripheral scan chain is selected through the JTAG port.

3) For each test pattern, the following sequence is performed:

   a) Stimuli are applied through available primary inputs (primary inputs of the 'C2700B0 cDSP device provide additional control and visibility).

   b) Data is shifted into the peripheral scan chain.

c) Test data is captured.

d) The scan chain contents are shifted out.

e) Response is read at available primary outputs.

In steps 3b and 3d, the scan shift in and shift out are done simultaneously.

5.3.2 Slave Mode Considerations

If any of your test methodologies require the core to be turned off, you must put the ’C2700B0 cDSP core in slave mode. Putting the core in slave mode ensures that the core is inactive. This prevents any type of interference to your test from the core input signals.

Typically, the core is put into slave mode when there is a need to run a test on another part of a cDSP device, such as the user logic. In such instances, the ’C2700B0 core must be completely inactive to prevent the test from being affected by core input signals. An example of such a test is the PMT test shown in Figure 5–3. See the *Submicron ASIC Products Design for Testability Reference Guide* for more information about PMT test.

Figure 5–4 shows an additional illustration of slave mode operation with ’C2700B0 emulation1 (E1).

Figure 5–3. PMT Testing of User Logic

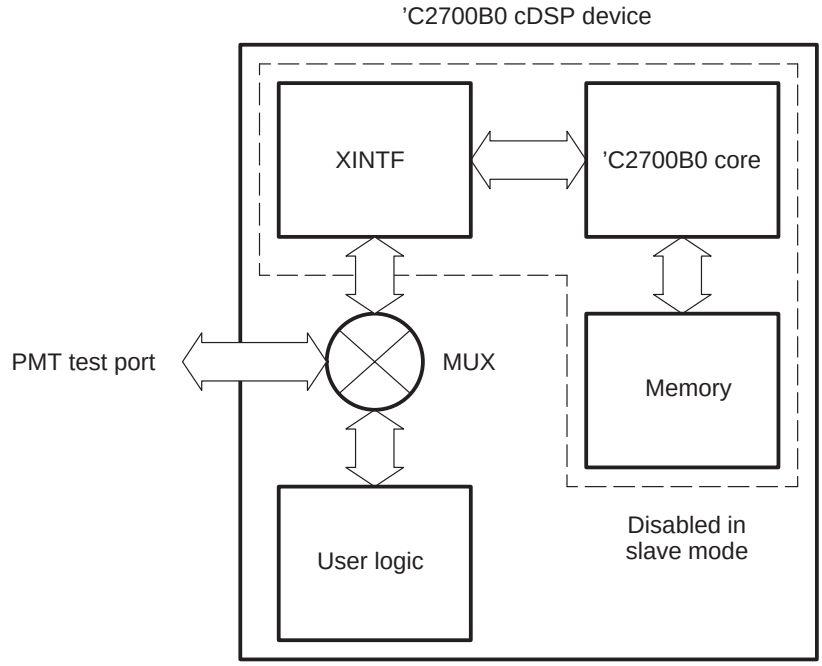
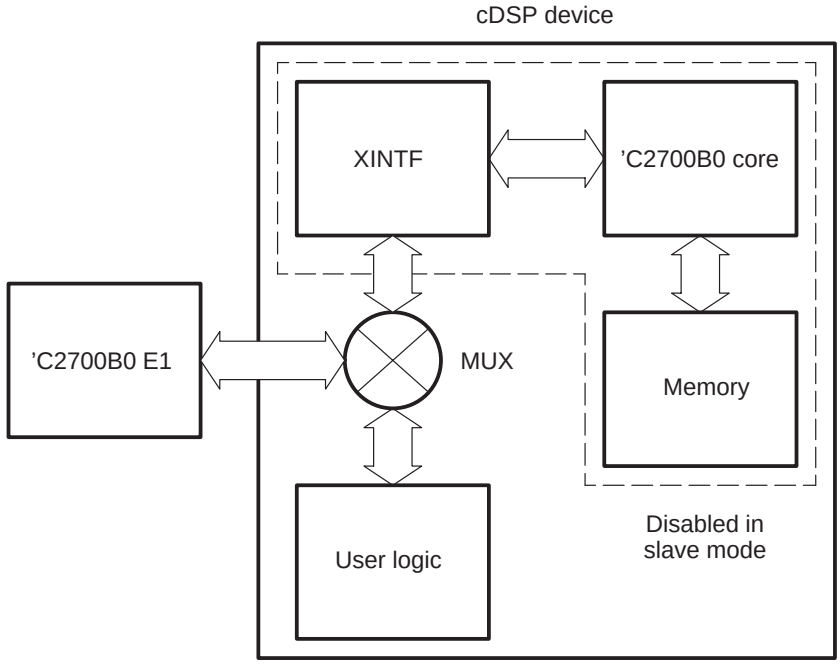


Figure 5–4. Slave Mode Operation With ’C2700B0 Emulation1



Putting the ’C2700B0 cDSP core in slave mode ensures that the core is disabled and does not respond to any inputs. The only signals that may affect the core in this mode are the control signals used for enabling and disabling slave mode operation. These control signals are the SLAVEIN, $\overline{\text{TRST}}$, ET0I, and ET1I signals.

Slave mode is activated within the core in one of the following ways:

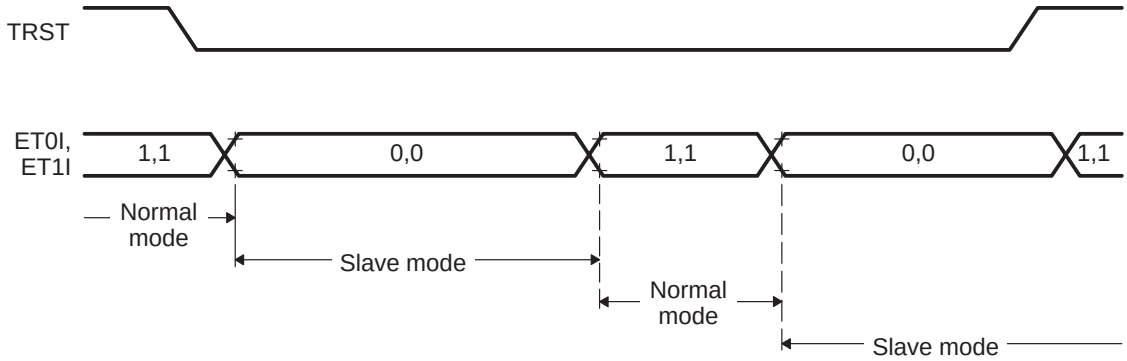
1) When the ’C2700B0 cDSP core contains particular combinations of ET0I, ET1I, and TRST input signals.

   Two specific combinations of the ET0I, ET1I, and TRST input signals activate slave mode (see Figure 5–5). These are:

   - When TRST is low and ET0I equals 0 and ET1I equals 0
   - When TRST is high after the rising edge of TRST-latched ET0I equals 0 and ET1I equals 0 (with this combination, slave mode is disabled only when TRST goes low and a different combination of ET0I and ET1I is observed)

   The combinational decode of ET0I, ET1I, and TRST inside the core activates the slave mode. Use this method to activate slave mode when you do not want to use a dedicated device pin for this purpose.

Figure 5–5. Slave Mode Activation



2) When the dedicated core input signal, SLAVEIN, is driven active high.

   When the SLAVEIN signal is active high, the ’C2700B0 CPU enters into slave mode and remains in this mode until SLAVEIN goes low.

   Use this activation method if your application requires a dedicated core input signal to activate slave mode from a device pin. When using this method of slave mode activation, the user logic must not generate a SLAVEIN signal. Instead, SLAVEIN must be directly pinned out.

Activation of slave mode automatically shuts off all the clocks, including the internal TCK logic and the system clock, SYSCLKOUT.

When the core is in slave mode, invalid values may appear in the core outputs (invalid values appear at random). Invalid values can cause the user logic to behave in an unpredictable manner. To protect the user logic from invalid core outputs, the output signal, XLOGOFF, is generated by the core. XLOGOFF, when active high, indicates to the user logic that the core is in an unknown state. At this point, any outputs from the CPU must be ignored by the components that connect to the core (a process known as isolation of the user logic from the core). Moreover, the output drivers of the components that drive the data read buses, PRDB and DRDB, must be in high impedance mode. The isolation of the user logic is illustrated in Figure 5–6 on page 5-14.

The output signals XLOGOFF, when active high, and COREATPG, when low, indicate that the ’C2700B0 core is in slave mode.

CAUTION

Changing the SLAVEIN input signal during the course of a test can be harmful to the test.

5.3.3 Considerations in Connecting User Logic to Memory Interface

The user logic is typically connected to TI-defined interface bridges, such as the XINTF. For more information, see Chapter 3, *External Interface (XINTF)*. In a typical connection, you do not need to make considerations for signals that are generated when the ’C2700B0 core is put under various test conditions. However, if any of your components is connected directly to the memory interface you must account for the following signals:

- **$\overline{\text{SYSRS}}$.** This is an output signal from the CPU that indicates the CPU is in a reset state.
- **ABORTREADY.** This is an output signal from the CPU that indicates all pending and currently active memory requests will be aborted.
- **XLOGOFF.** This is an output signal from the CPU that indicates the current CPU outputs contain invalid values. This means that all components use 3-state outputs on the data read buses, PRDB and DRDB, and ignore any memory interface signals.
- **MEMXFTEST.** This is an output signal from the CPU that indicates a functional test of the memory interface components is being performed so that all dynamic memory mapping operations will be turned off.

These signals are generated when the ’C2700B0 core is put under various test conditions. You must account for these signals according to the following criteria:

- During test, treat XLOGOFF as the highest priority.

  Therefore, when XLOGOFF is high you must ensure that the data read buses, PRDB and DRDB, are in asynchronous 3-state.

- Treat ABORTREADY and SYSRSn as the next highest priority.

  When these signals appear, you must ensure that all pending requests are cleared. In addition, you must ensure that write buffers are flushed and state machines in the memory wrappers or interface bridges are reset to their initialization state.

Table 5–2 summarizes design considerations that must be addressed for various output states of the signals listed above. For debug and emulation purposes, you must design the logic so that it complies to the behavior described in Table 5–2.

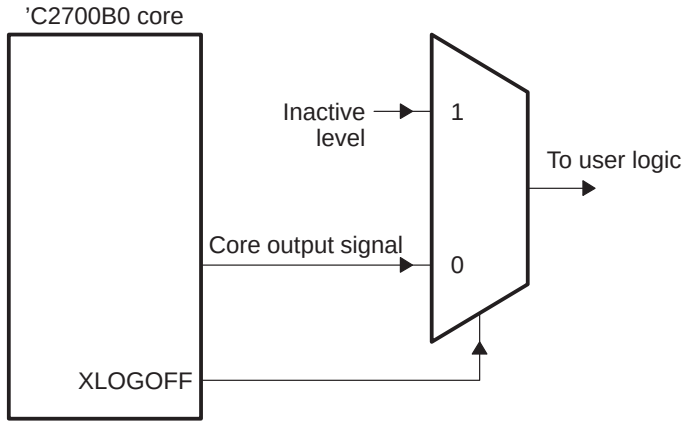
Table 5–2. Design Considerations for Various Signal Output States

Signal Output States	Design Considerations
When XLOGOFF=1	1) You must ensure that the drivers that drive PRDB and DRDB are put into the high-impedance state asynchronously. 2) You must reset all the internal state machines that are used for arbitration, wait states, etc. POREADY, DROREADY, and DWOREADY signals (whichever is connected) must be driven high. Ignore all memory interface signals when XLOGOFF is high. 3) You must ensure that PRDY or DRDY (whichever is connected) is driven low. Operations 2 and 3 must be done synchronously with SYSCLKOUT.
XLOGOFF=0 and ABORTREADY=1 or SYSRSN=0	1) You must reset all the internal state machines that are used for arbitration, wait states, etc. POREADY, DROREADY, and DWOREADY signals (whichever is connected) must be driven high. Ignore all memory interface components when ABORTREADY=0 and SYSRSn=1. 2) You must ensure that PRDY or DRDY (whichever is connected) is driven low. Operations 1 and 2 must be done synchronously with SYSCLKOUT.
XLOGOFF=0 ABORTREADY=0 SYSRSN=1	This is the normal mode of operation. Memory interface signals must be decoded and requested memory operations must be performed.
MEMXFTEST=1	This is optional and needs to be used only if there is dynamic memory mapping. For example, consider a ROM that is mapped to program space. The MPNMC signal, which is generated by the user logic, is used to turn the ROM off in the memory space. This allows the MPNMC signal to dynamically map the ROM on and off the memory space. In such a case, the MEMXFTEST signal must be used to override the MPNMC signal during functional test of the memory interface components.

5.3.4 Considerations in Isolating the User Logic

Table 5–2 describes the conditions under which the external logic isolates itself from the ’C2700B0 cDSP core. During isolation, the memory interface components are internally shut off. Isolation of the external logic ensures that the memory interface components are not affected by invalid CPU outputs. Isolation is achieved by gating off the core output signals. The isolation of the user logic is illustrated in Figure 5–6.

Figure 5–6. Isolation of User Logic From Core Outputs



The following is a list of the core output signals that are used for making memory requests:

- PRDS0
- PRDS1
- PWDS0
- PWDS1
- DRDS0
- DRDS1
- DRLSB
- DRMSB
- DWDS0
- DWDS1
- DWLSB
- DWMSB
- IACK

Since memory interface components use these signals to decode CPU requests, you can gate off the signals internally to achieve isolation.

Chapter 6

T320C2700B0 cDSP Design Example

This chapter builds on the information introduced in previous chapters by describing the generation and simulation of an example ’C2700B0 cDSP design.

Topic **Page**

6.1 Overview

The sections in this chapter describe the generation of a ’C2700B0 design according to the following sequence of events:

1) A top-level VHDL netlist is first created for the desired design memory map.

2) A ’C2700B0 assembly language test program is written, assembled, converted to simulator ROM format, and stored in a program ROM.

3) A top-level test bench is written to test the design by properly initializing the system that executes the assembled test program.

4) The design is synthesized, and the synthesis output is resimulated to ensure proper functionality.

Figure 6–1 shows the memory map for the example design. The minimum B0 and B1 SARAM requirements are met and a ROM exists in the top memory location where the reset and interrupt vector table and the test program are stored. An external interface (XINTF) occupies the remaining memory space. An off-chip memory is used as an example peripheral to test the functionality of the XINTF.

Figure 6–2 shows the connections for the top-level design. For simplicity, an ATPG scan chain has been omitted from this example. See Figure 5–2 on page 5-8 for information on how to incorporate a scan chain into your design.

Figure 6–1. T320C2700B0 cDSP Example Design Memory Map

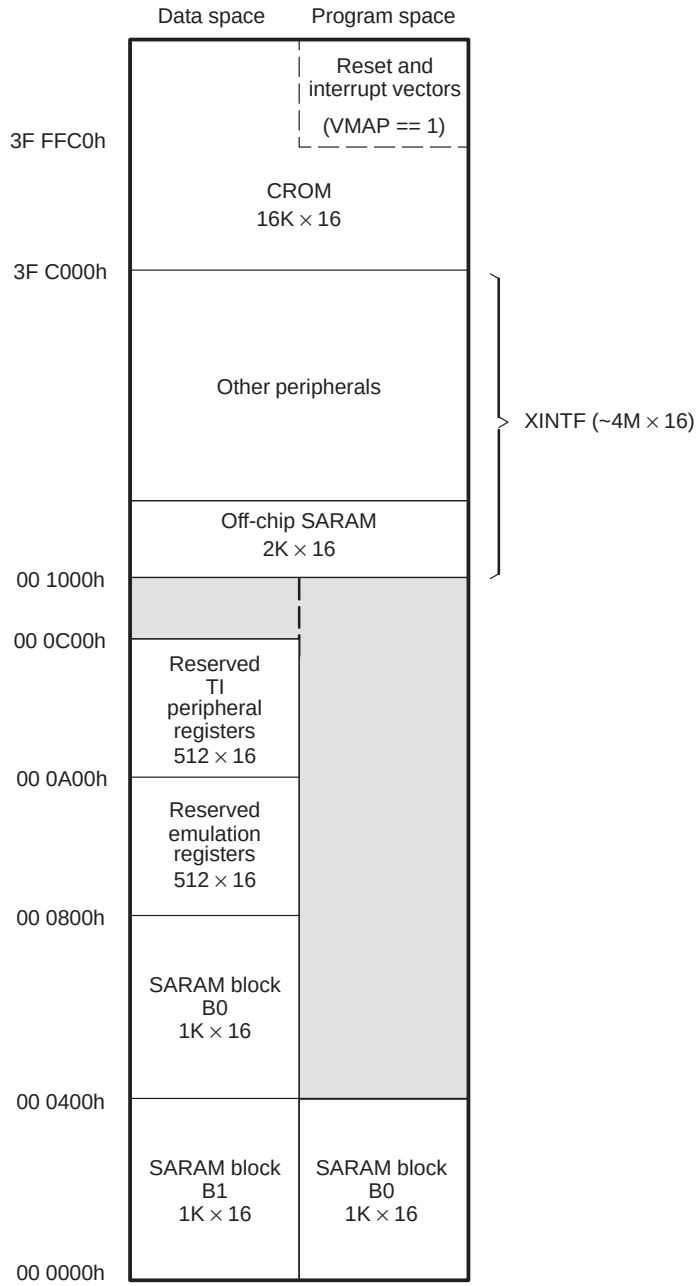
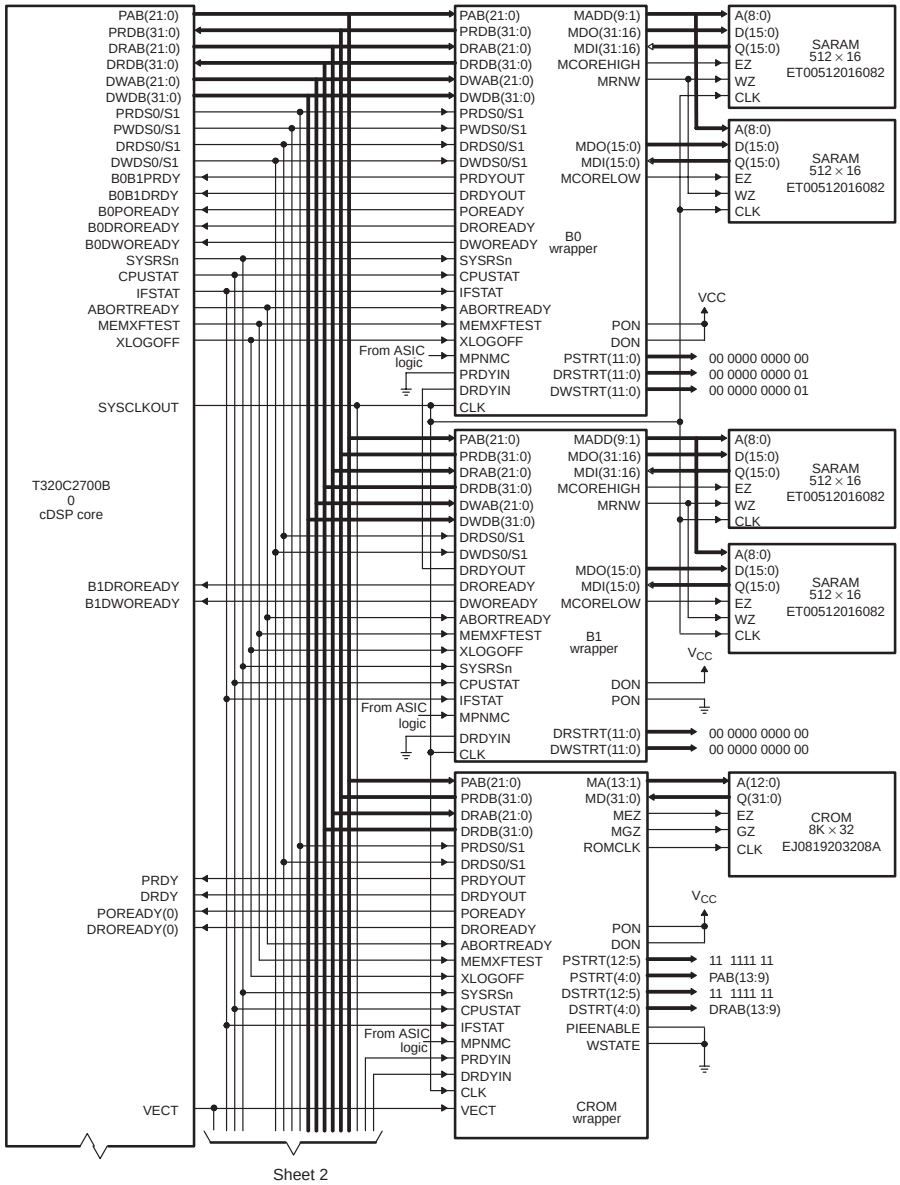
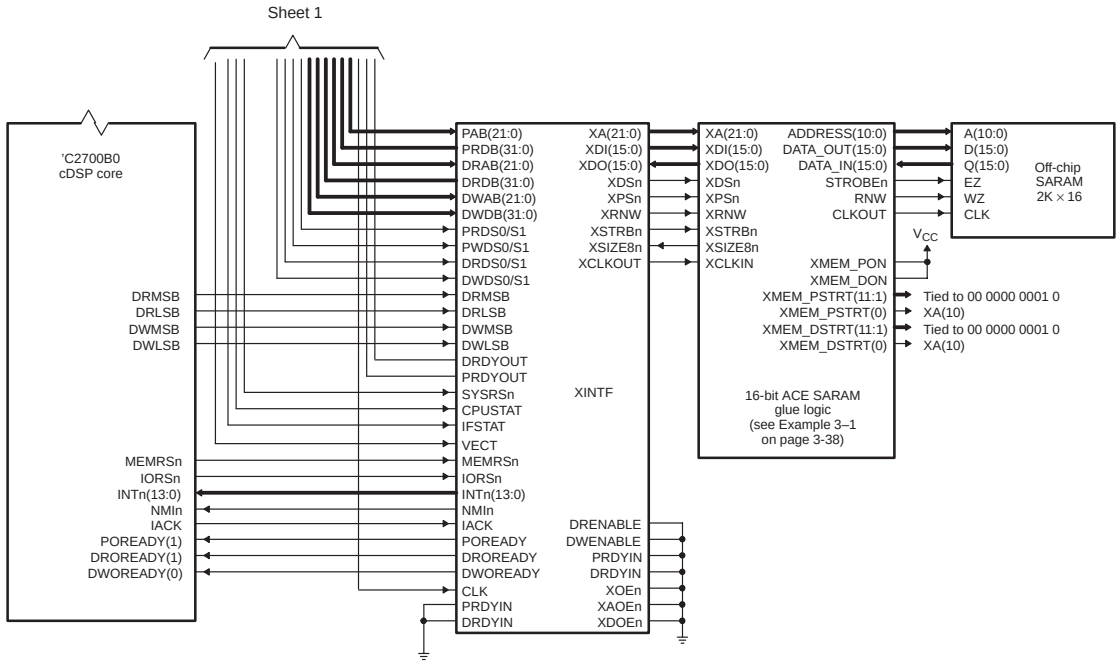


Figure 6–2. T320C2700B0 cDSP Example Design Configuration



Sheet 2

Figure 6–2. T320C2700B0 cDSP Example Design Configuration (Continued)



Note: SYSCLKOUT is not connected directly to CLK on the XINTF and memory wrappers, SYSCLKOUT must go through a clock tree synthesis (CTS) buffer.

6.2 Generating a Top-Level VHDL Netlist

Once the design requirements are realized, the necessary VHDL megamodules must be collected into a working design directory. The ’C2700B0 design kit contains the SARAM wrapper, CROM wrapper, and XINTF all in register-transfer-level (RTL) VHDL. The memory cores can be generated using the Texas Instruments Design Support Software (TIDSS) tools. If you do not have these tools locally installed, TI can produce the needed memory core models. A behavioral C model of the ’C2700B0 core is provided in the design kit for simulation. Any external peripherals, glue logic, or miscellaneous items must also be included.

A top-level model must now be created using the configuration rules presented in this book. DSPnetGEN is a very useful tool for creating a netlist for the ’C2700B0 core, memory, and XINTF parts of a design. See Appendix C for a detailed tutorial on how to use this tool. DSPnetGEN eliminates most of the tedious connection issues involved with a complex design. A few additions must be made manually to the DSPnetGEN output, such as attaching external peripherals/glue logic to the XINTF, adding a phase-locked loop (PLL) to the design, or adding a multiplexer to complete a scan chain. DSPnetGEN uses golden netlist file (GNF) files to determine how to connect the modules together. The ’C2700B0 GNF files have already been developed and are included in the design kit.

The example top-level VHDL code is too extensive to present in this book, but it is summarized by the connection diagram in Figure 6–2.

6.3 Generating a TMSC2700B0 Assembly Language Test Program

Once the top-level netlist for a design has been generated, a test program must be written to verify that every aspect of the system is functioning properly. For this example, a self-address memory write/read test has been written to test B0, B1, and the external RAM (the address number is the data written to that address). The ROM is tested for read functionality, since the test program is read from it. The purpose of this test is simply to verify every component in the system is working properly. More exhaustive memory tests should be used when testing an actual physical device.

Example 6–1 shows the example test program. For a ’C2700B0 instruction set summary, see Appendix B. For a detailed explanation of the ’C2700B0 instruction set, see *TMS320C27xx DSP CPU and Instruction Set Reference Guide* and for the assembly language tools, see *TMS320C27xx Assembly Language Tools User’s Guide*.

Assemble the program to produce an object file. A command file is then needed with this object file to provide the linker information about the system memory map and where in the TMS320C27xx simulator memory to load the assembled program. The linker produces a common object file format (COFF) file that can then be loaded into the simulator. Example 6–2, page 6-10, shows the example command file.

Example 6–3, page 6-10, shows a script that generates an object file, COFF file, absolute list file, and hexadecimal assembled code file. These executables, as well as the simulator and emulator, are available in the TMS320C27xx tools package.

Before proceeding, load the COFF file into the TMS320C27xx simulator to ensure proper performance of the assembly code. The help files included with the simulator will teach you how to use this tool. In the case of our example, the program counter begins at 3F C000h, the beginning of the ROM where you loaded the program.

Example 6–1. T320C2700B0 cDSP Example Design Assembly Language Test Program

```
*******************************************************************
*******************************************************************
***
*** This program will be loaded in an internal ROM at 0x3FC000.
*** The reset and interrupt vector table will be loaded at 0x3FFFC0
*** through the VHDL code, and the reset vector will point to
*** 0x3FC000.  The program tests read/write functionality for a
*** ’C2700B0 core, internal ROM, B0 SARAM, B1 SARAM, XINTF, and
*** off–chip RAM setup.
***
*******************************************************************
*******************************************************************

B0DSTRT   .set   0x000400
B0DEND    .set   0x000800

B1DSTRT   .set   0x000000
B1DEND    .set   0x000400

EXRAMS    .set   0x001000
EXRAME    .set   0x001400

          .sect  ”Code”

*** Write 32–bit self–addresses to B1 block ***

SELFWB1   NOP    *ARP7
          MOV    XAR6,#B1DEND
          MOV    XAR7,#B1DSTRT
          MOVL   ACC,@XAR7
LOOP1     MOVL   *++,ACC
          ADD    ACC,#2
          CMPL   ACC,@XAR6
          B      LOOP1,NEQ
ENDSWB1   NOP

*** Read B1 self–address pattern ***

SELFRB1   NOP    *ARP7
          MOV    XAR6,#B1DEND
          MOV    XAR7,#B1DSTRT
LOOP2     MOVL   ACC,*++
          MOVL   ACC,@XAR7
          CMPL   ACC,@XAR6
          B      LOOP2,NEQ
ENDSRB1   NOP
```

Example 6–1. T320C2700B0 cDSP Example Design Assembly Language Test Program (Continued)

```
*** Write 32-bit self-addresses to B0 block ***

SELFWB0   NOP    *ARP7
          MOV    XAR6,#B0DEND
          MOV    XAR7,#B0DSTRT
          MOVL   ACC,@XAR7
LOOP3     MOVL   *++,ACC
          ADD    ACC,#2
          CMPL   ACC,@XAR6
          B      LOOP3,NEQ
ENDSWB0   NOP

*** Read B0 self-address pattern ***

SELFRB0   NOP    *ARP7
          MOV    XAR6,#B0DEND
          MOV    XAR7,#B0DSTRT
LOOP4     MOVL   ACC,*++
          MOVL   ACC,@XAR7
          CMPL   ACC,@XAR6
          B  LOOP4,NEQ
ENDSRB0   NOP

*** Write 16-bit self-addresses to External RAM ***

SELFWXR   NOP    *ARP7
          MOV    XAR6,#EXRAME
          MOV    XAR7,#EXRAMS
          MOVL   ACC,@XAR7
LOOP5     MOV    *++,ACC
          ADD    ACC,#1
          CMPL   ACC,@XAR6
          B      LOOP5,NEQ
ENDSWXR   NOP

*** Read External RAM self-address pattern ***

SELFRXR   NOP    *ARP7
          MOV    XAR6,#EXRAME
          MOV    XAR7,#EXRAMS
LOOP6     MOV    ACC,*++
          MOV    ACC,@XAR7
          CMPL   ACC,@XAR6
          B      LOOP6,NEQ
ENDSRXR   NOP

       .end
```

Example 6–2. T320C2700B0 cDSP Example Design Assembly Language Command File

```
MEMORY
{
   PAGE 0 :  B0PRAM:   origin = 0000h,     length = 0400h
             PEXRAM:   origin = 1000h,     length = 0800h
             PROM:     origin = 3FC000h,   length = 4000h
   PAGE 1 :  B1DRAM:   origin = 0000h,     length = 0400h
             BODRAM:   origin = 0400h,     length = 0400h
             DEXRAM:   origin = 1000h,     length = 0800h
             DROM:     origin = 3FC000h,   length = 4000h
}

SECTIONS
{
   Code: {} >  PROM PAGE 0
}
```

Example 6–3. T320C2700B0 cDSP Example Design Assembly Language File Generation Script

```
#!/bin/csh –f
asm27 –s $1
lnk27 –o $1.coff –m $1.map $1.cmd $1.obj
abs27 $1.coff
asm27 –a $1.abs
hex27 –a $1.coff –o $1.hex –memwidth $2 –romwidth $2
```

6.4 Programming a VHDL ROM for Simulation

When the assembly test program is working correctly, the next step is to convert the hexadecimal assembled code (the output of the *hex27* utility) into binary ROM format for the VHDL simulation. Example 6–4 shows the hexadecimal assembled code file. The file is already set up for 32-bit wide memories because of the *memwidth* and *romwidth* options used with the *hex27* utility. A Perl script for converting hexadecimal assembled code (Example 6–4) to ROM format (Example 6–5) is shown in Example 6–6.

Once the ROM binary ASCII file is created, the VHDL file I/O can be used to program the VHDL ROM 32 bits at a time. When a VHDL ROM model is generated with the TIDSS ACE toolkit, an additional ROM program file template accompanies it. Example 6–7, page 6-14, shows an example VHDL ROM program file. The ROM architecture in the main ROM VHDL file makes a call to the function EJ0819203208A_program that returns a variable of type RETURN_TYPE in order to program the ROM. Different methods can be used within the function to program the ROM as long as the function returns a variable of type RETURN_TYPE. In Example 6–7, the reset and interrupt vector table is loaded in the upper 32 × 32 bits of the ROM (starting at 3F FFC0h). The table could have instead been loaded into the ROM earlier in the simulation process through use of a separate vector table assembly code section. All the interrupt vectors point to the reset vector because no interrupts are used in this simulation.

The top-level design can now be compiled to check for any syntax errors. Example 6–8, page 6-17, shows a script that can be set up to compile the entire design for a QuickHDL simulation with one command. The order of compilation remains unchanged for different simulators.

Example 6–4. Hexadecimal Assembled Code

```
00 00 76 1F 76 80 77 B7 76 C0 04 00 06 A7 00 00 09 02 1E B9 60 FD 0F A6
77 B7 77 00 04 00 76 80 00 00 76 C0 06 A7 06 B9 60 FD 0F A6 76 1F 77 00
77 B7 00 00 08 00 76 80 04 00 76 C0 1E B9 06 A7 0F A6 09 02 77 00 60 FD
76 80 77 B7 76 C0 08 00 06 B9 04 00 0F A6 06 A7 77 00 60 FD 00 00 76 1F
76 80 77 B7 76 C0 14 00 06 A7 10 00 09 02 1E B9 60 FD 0F A6 77 B7 77 00
14 00 76 80 10 00 76 C0 06 A7 06 B9 60 FD 0F A6 00 00 77 00
```

Example 6–5. 32-Bit Wide Binary ROM Output of Hexadecimal Assembled Code

```
00000000000000000111011000011111
01110110100000000111011110110111
01110110110000000000010000000000
00000110101001110000000000000000
00001001000000100001111010111001
01100000111111010000111110100110
01110111101101110111011100000000
00000100000000000011101101000000
00000000000000000011101101000000
00000110101001110000011010111001
01100000111111010000111110100110
01110110000111110111011100000000
01110111101101110000000000000000
00001000000000000011101101000000
00000100000000000011101101000000
00011110101110010000011010100111
00001111101001100000100100000010
01110111000000000110000011111101
01110110100000000111011110110111
01110110110000000000100000000000
00000110101110010000010000000000
00001111101001100000011010100111
01110111000000000110000011111101
00000000000000000011101000011111
01110110100000000111011110110111
01110110110000000000101000000000
00000110101001110001000000000000
00001001000000100001111010111001
01100000111111010000111110100110
01110111101101110111011100000000
00010100000000000011101101000000
00010000000000000011101101000000
00000110101001110000011010111001
01100000111111010000111110100110
00000000000000000011101100000000
```

Example 6–6. Perl Script for Converting Hexadecimal Assembled Code to ROM Format

```
#!/usr/local/bin/perl
#
# Converts hex files to binary ROM format of specified width (8–bit minimum)
#
# Sample script:
#    hex27 –a test.coff –o test.hex –memwidth 32 –romwidth 32
#    hex2rom.pl test.hex test.rom 32

if (@ARGV != 3)
{
   print ”\nWrong number of arguments.”;
   print ”\nProper usage:  hex2rom.pl infilename outfilename romwidth\n\n”;
   exit(1);
}

open (IN, ”@ARGV[0]”) || die ”\nCan’t open file @ARGV[0]\n\n”;
open (OUT, ”>@ARGV[1]”);

@hexline = <IN>;

# Remove comments inserted by hex27 utility
pop(@hexline);
shift(@hexline);
shift(@hexline);

chop(@hexline);

for($i=0; $i<@hexline; $i++)
{
   @temp = split(/ /, $hexline[$i]);
   for($j=0; $j<@temp; $j++)
   {
      push(@hex8, $temp[$j]);
   }
}

for($i=0; $i<@hex8; $i++)
{
   for ($j=7; $j>=0; $j––)
   {
      $temp2 = (hex($hex8[$i]) >> $j) & 01;
      print OUT $temp2;
   }
   if (!(($i+1) % (@ARGV[2]/8)))
   {
      print OUT ”\n”;
   }
}

close(IN);
close(OUT);

print ”\n@ARGV[0] was converted to @ARGV[2]–bit binary ROM format and stored as
@ARGV[1]\n\n”;
```

Example 6–7. ROM VHDL Program File

```
library IEEE;
 use IEEE.STD_LOGIC_1164.all;
 use IEEE.STD_LOGIC_UNSIGNED.all;
 library STD;
 use STD.TEXTIO.ALL;

 package EJ0819203208A_prog is
   constant WORD_NUMBER : integer := 8192;
   constant WORD_SIZE : integer := 32;

-- Reset and interrupt vector table

 constant RESET   : std_logic_vector(31 downto 0) := "00000000001111111100000000000000";
 constant INT1    : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT2    : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT3    : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT4    : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT5    : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT6    : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT7    : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT8    : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT9    : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT10   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT11   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT12   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT13   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant INT14   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant DLOGINT : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant RTOSINT : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant EMUINT  : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant NMI     : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant ILLEGAL : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant USER1   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant USER2   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant USER3   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant USER4   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant USER5   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant USER6   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant USER7   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant USER8   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant USER9   : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant USER10  : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant USER11  : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";
 constant USER12  : std_logic_vector(31 downto 0) := "00000000001111111111111111111000";

 type RETURN_TYPE is array (0 to (WORD_NUMBER – 1)) of
      std_logic_vector(0 to (WORD_SIZE–1));

 function EJ0819203208A_program return RETURN_TYPE;

end package EJ0819203208A_prog;
```

Example 6–7.ROM VHDL Program File (Continued)

```
 package body EJ0819203208A_prog is

   function  EJ0819203208A_program
      return RETURN_TYPE is

      variable TEMP : RETURN_TYPE;
      variable I : integer range 0 to (WORD_NUMBER – 1) := 0;
      variable ENTRY : bit_vector(0 to 31);
      variable TEMP2 : integer;
      variable FILE_LINE : line;
      variable CONVERT : std_logic_vector(0 to 31);

      file INFILE : text is in ”test_program.rom”;

   begin

-- Load test program at start of ROM

      TEMP2 := WORD_NUMBER – 32;

      while ((not endfile(INFILE)) and (I < TEMP2)) loop
         readline (INFILE, FILE_LINE);
         read (FILE_LINE, ENTRY);
         for J in 0 to 31 loop
            if (ENTRY(J) = ’0’) then
               TEMP(I)(J) := ’0’;
            elsif (ENTRY(J) = ’1’) then
               TEMP(I)(J) := ’1’;
            else
               TEMP(I)(J) := ’U’;
            end if;
         end loop;
         I := I + 1;
      end loop;

-- Initialize unused memory to 0

      while (I < TEMP2) loop
         TEMP(I) := ”00000000000000000000000000000000”;
         I := I + 1;

      end loop;
```

Example 6–7. ROM VHDL Program File (Continued)

```
-- Load reset and interrupt vector table at upper 32x32 of ROM (starting at 0x3FFFC0)

   TEMP(TEMP2) := RESET;
   TEMP(TEMP2 + 1) := INT1;
   TEMP(TEMP2 + 2) := INT2;
   TEMP(TEMP2 + 3) := INT3;
   TEMP(TEMP2 + 4) := INT4;
   TEMP(TEMP2 + 5) := INT5;
   TEMP(TEMP2 + 6) := INT6;
   TEMP(TEMP2 + 7) := INT7;
   TEMP(TEMP2 + 8) := INT8;
   TEMP(TEMP2 + 9) := INT9;
   TEMP(TEMP2 + 10) := INT10;
   TEMP(TEMP2 + 11) := INT11;
   TEMP(TEMP2 + 12) := INT12;
   TEMP(TEMP2 + 13) := INT13;
   TEMP(TEMP2 + 14) := INT14;
   TEMP(TEMP2 + 15) := DLOGINT;
   TEMP(TEMP2 + 16) := RTOSINT;
   TEMP(TEMP2 + 17) := EMUINT;
   TEMP(TEMP2 + 18) := NMI;
   TEMP(TEMP2 + 19) := ILLEGAL;
   TEMP(TEMP2 + 20) := USER1;
   TEMP(TEMP2 + 21) := USER2;
   TEMP(TEMP2 + 22) := USER3;
   TEMP(TEMP2 + 23) := USER4;
   TEMP(TEMP2 + 24) := USER5;
   TEMP(TEMP2 + 25) := USER6;
   TEMP(TEMP2 + 26) := USER7;
   TEMP(TEMP2 + 27) := USER8;
   TEMP(TEMP2 + 28) := USER9;
   TEMP(TEMP2 + 29) := USER10;
   TEMP(TEMP2 + 30) := USER11;
   TEMP(TEMP2 + 31) := USER12;

   return TEMP;

   end function EJ0819203208A_program;

 end package body EJ0819203208A_prog;
```

Example 6–8. QuickHDL VHDL Compilation Script

```
### Compile DW libraries needed for XINTF

qvhcom –work DW01 \
dw01_cmp2.ent \
dw01_cmp2_sim.vhd

qvhcom –work DW03 \
dw03_updn_ctr.ent \
dw03_updn_ctr_sim.vhd


### Compile XINTF sub–module entities

qvhcom \
xclock.ent \
xcntrl.ent \
xdecoder.ent \
xemureg.ent \
xfifo.ent \
xintfil.ent \
xintsync.ent \
xlatch.ent \
xperreg.ent \
xrwblk.ent \
xwgen.ent


### Compile XINTF sub–module RTL

qvhcom –explicit \
xclock.rtl \
xcntrl.rtl \
xdecoder.rtl \
xemureg.rtl \
xfifo.rtl \
xintfil.rtl \
xintsync.rtl \
xlatch.rtl \
xperreg.rtl \
xrwblk.rtl \
xwgen.rtl


### Compile XINTF top–level module

qvhcom \
xintf.ent \
xintf.rtl


### Compile external memory core and external memory glue logic

qvhcom –explicit \
ET02048016042.vhd \
xmem_glue_logic.vhd
```

Example 6–8. QuickHDL VHDL Compilation Script (Continued)

```
### Compile CROM wrapper

qvhcom \
cromw.ent \
cromw.rtl

### Compile SARAM wrapper

qvhcom \
sramwagen.ent \
sramwdec.ent \
sramwfsm.ent \
sramwagen.rtl \
sramwdec.rtl \
sramwfsm.rtl \
sramw.ent \
sramw.rtl


### Compile T320C2700B0 core

qvhcom \
ti_bus_timing.vhd \
watchblock.vhd \
SC_ANKOOR_C3.vhd \
T320C2700func.vhd \
T320C2700B0.vhd


### Compile ROM program file

qvhcom –explicit –work TI_CUSTOM \
EJ0819203208A_prog.vhd


### Compile internal memories

qvhcom –explicit \
EJ0819203208A.vhd \
ET00512016082.vhd


### Compile top–level design file and testbench (once written)

qvhcom \
example_design.vhd \
test_example_design.vhd
```

6.5 Generating a Top-Level Test bench for Simulation

A top-level test bench must be written to properly initialize the system for simulation. Table 6–1 summarizes the signals that must be initialized to a specific value in order for the example system to function properly. The signals that are not accessible from the top level can be initialized by the simulator. In a more complex design, some of these signals, such as the interrupt signals, scan path signals, or the emulator controlled registers in the XINTF, are controlled by other parts of the system and do not need to be hard wired. Once the appropriate signals are tied off, the simulation starts by driving reset low until the system stabilizes. When reset is brought high again, a program read request to 3F FFC0h (reset vector) occurs if VMAP equals 1 or to 00 0000h if VMAP equals 0. The reset and interrupt vector table is located at 3F FFC0h in this example design; therefore, VMAP is set to 1.

Table 6–1. T320C2700B0 cDSP Example Design Default Signal Values for System Initialization

Signal	Default Value
t320c2700i0/nmin	1
t320c2700i0/int14n	1
t320c2700i0/int13n	1
t320c2700i0/int12n	1
t320c2700i0/int11n	1
t320c2700i0/int10n	1
t320c2700i0/int9n	1
t320c2700i0/int8n	1
t320c2700i0/int7n	1
t320c2700i0/int6n	1
t320c2700i0/int5n	1
t320c2700i0/int4n	1
t320c2700i0/int3n	1
t320c2700i0/int2n	1
t320c2700i0/int1n	1
t320c2700i0/vmap	1
t320c2700i0/slavein	0
t320c2700i0/scout	0
t320c2700i0/scpath	0
t320c2700i0/extcount1	0

Table 6–1 T320C2700B0 cDSP Example Design Default Signal Values for System Initialization (Continued)

Signal	Default Value
t320c2700i0/breaktagl	0
t320c2700i0/exttrgr	0
t320c2700i0/et0i	1
t320c2700i0/rtosint	1
t320c2700i0/tracetagh	0
t320c2700i0/tms	0
t320c2700i0/tracetagl	0
t320c2700i0/tdi	0
t320c2700i0/trst	0
t320c2700i0/monpriv	0
t320c2700i0/et1i	1
t320c2700i0/breaktagh	0
t320c2700i0/extcount0	0
sramwi0/srwTWG2scin	0
sramwi0/srwTWG2scen	0
sramwi1/srwTWG2scin	0
sramwi1/srwTWG2scen	0
cromwi0/crwTWG2scin	0
cromwi0/crwTWG2scen	0
xintfi0/xifTWG2scin	0
xintfi0/xifTWG2scen	0
cromwi0/extNWG2mpnmc	0
cromwi0/cpuTWG2memxftest	1
sramwi0/cpuTWG2memxftest	1
sramwi0/extNWG2mpnmc	0
sramwi1/cpuTWG2memxftest	1
sramwi1/extNWG2mpnmc	0
xintfi0/perNBG2xintn	11 1111 1111 1111
xintfi0/perNWG2xnmin	1
xintfi0/perNWG2xoffn	1

Table 6–1 T320C2700B0 cDSP Example Design Default Signal Values for System Initialization (Continued)

Signal	Default Value
xintfi0/xifNWG2xdoen	0
xintfi0/u1/xcktbg2mode	00
xintfi0/u10/xprTBG2dtim0	0001 0000 0000 0001
xintfi0/u10/xprTBG2dtim1	0001 0000 0000 0001
xintfi0/u10/xprTBG2dtim2	0001 0000 0000 0001
xintfi0/u10/xprTBG2dtim3	0001 0000 0000 0001
xintfi0/u10/xprTBG2dtim4	0001 0000 0000 0001
xintfi0/u10/xprTBG2ptim0	0001 0000 0000 0001
xintfi0/u10/xprTBG2ptim1	0001 0000 0000 0001

6.6 Example Simulation Displays

Figure 6–3 through Figure 6–6 illustrate various stages of the design simulation. A successful reset occurs in Figure 6–3 as a program read is requested at the reset address 3F FFC0h. The reset vector points to the beginning of the ROM (3F C000h) and execution of the test program begins in Figure 6–4 with the first line of assembled code (0000 761Fh, see Example 6–4 on page 6-11). A successful write to B0 SARAM is shown in Figure 6–5 and a read from B0 SARAM is shown in Figure 6–6.

Figure 6–3. T320C2700B0 cDSP Example Design Simulation Display — System Reset

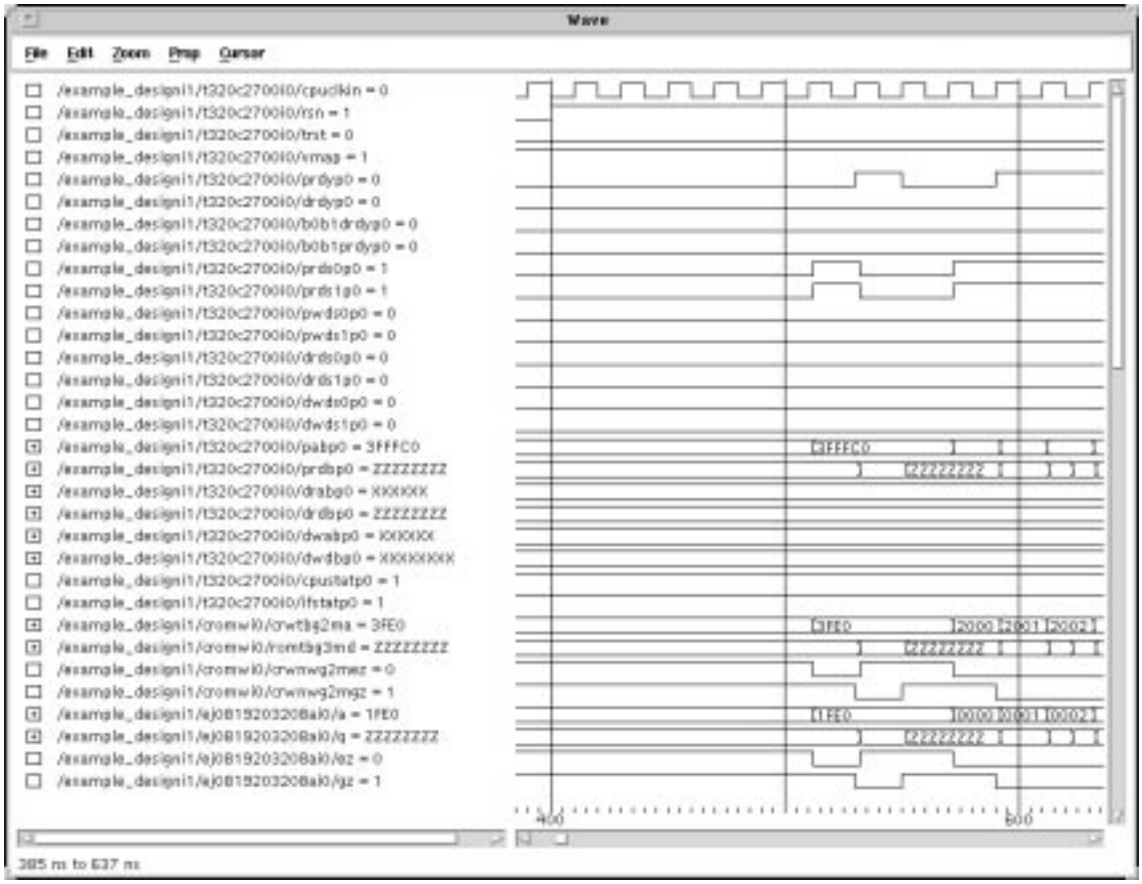


Figure 6–4. T320C2700B0 cDSP Example Design Simulation Display — Test Program Start

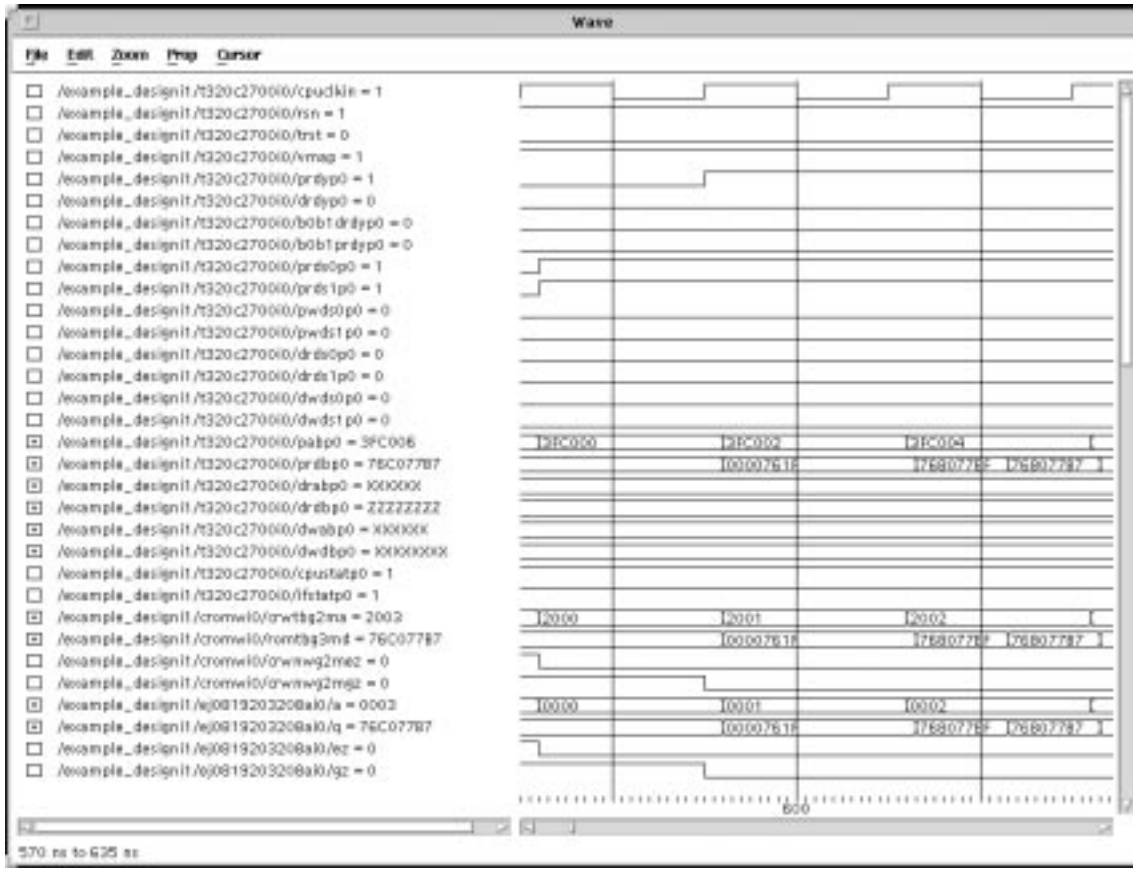


Figure 6–5. T320C2700B0 cDSP Example Design Simulation Display — B0 SARAM Write

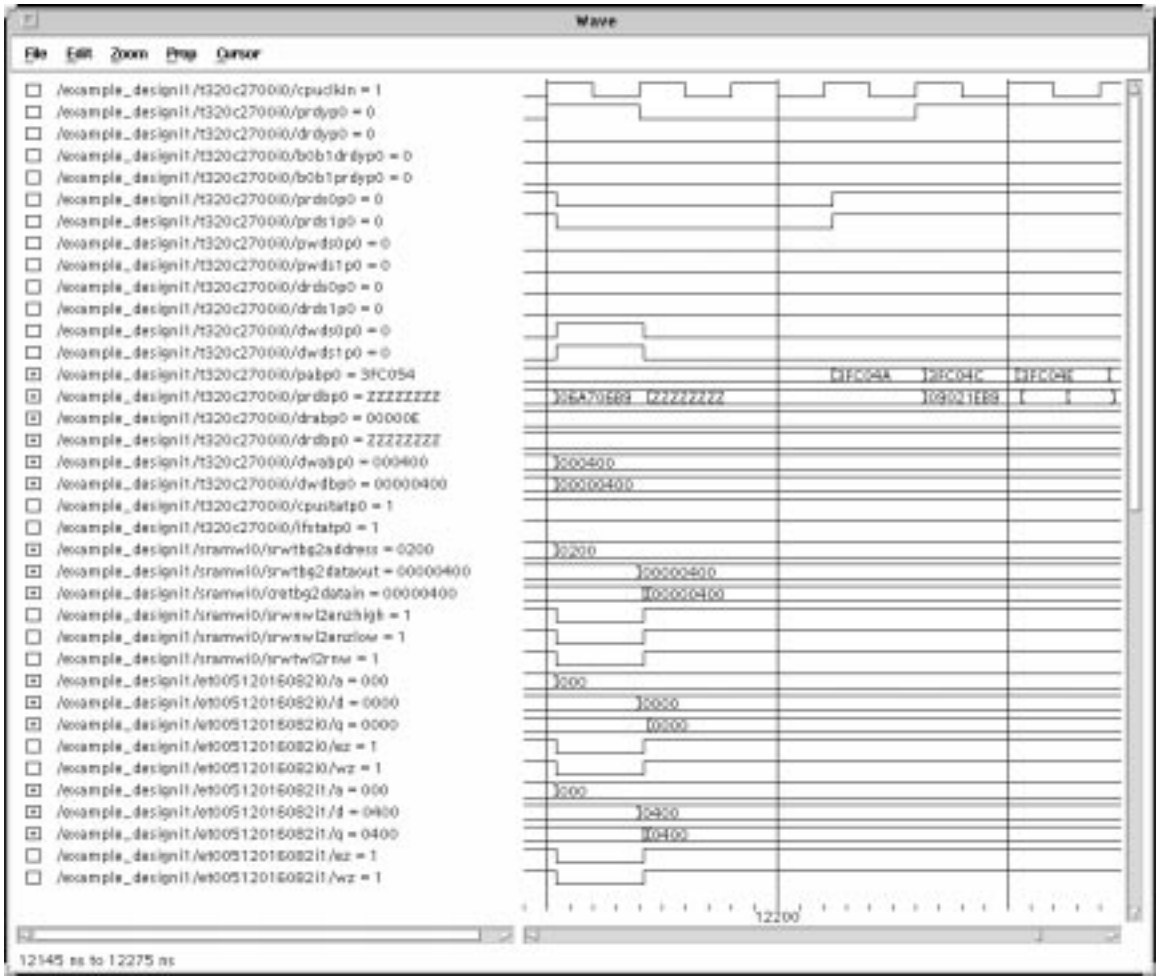
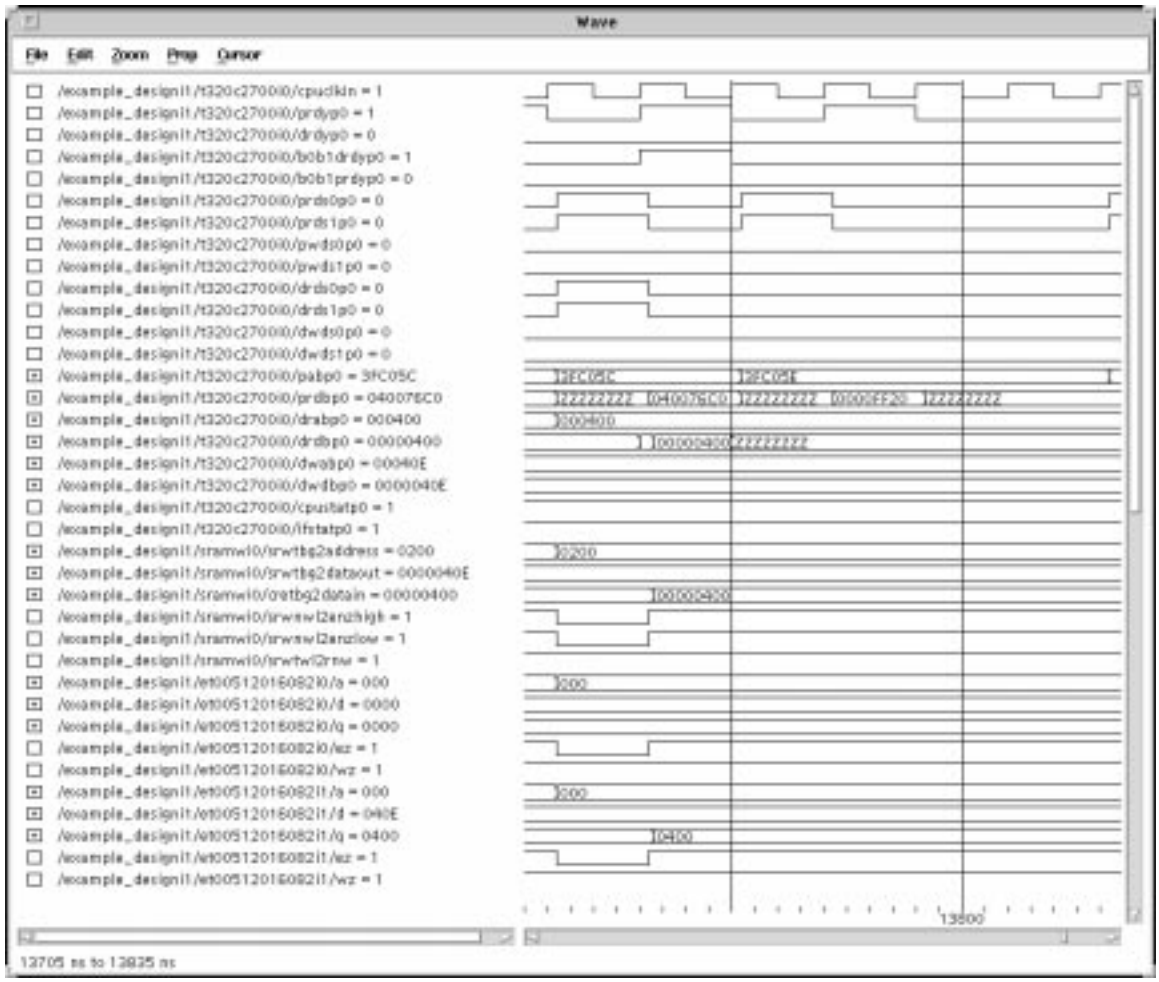


Figure 6–6. T320C2700B0 cDSP Example Design Simulation Display — B0 SARAM Read



6.7 Synthesizing Your Design

Once you are confident that your RTL design is simulating properly, you can synthesize your design. Synthesizing is the process of mapping generic RTL logic to a specific library and optimizing your design at the gate level. This section summarizes the synthesis process using Synopsys' Design Analyzer.

The ACE memory and 'C2700B0 models that were used for the RTL simulation are behavioral models and cannot be synthesized. You need new Synopsys models for these components to properly analyze the design. The 'C2700B0 model can be found in the design kit. ACE Toolkit has an option to output Synopsys memory models and symbols. The symbol output is a script that you run at the UNIX™ prompt to create a slib file. You can generate a sdb file from this slib file in a Synopsys dc_shell. The following is an example script to create a sdb symbol library:

```
read_lib ET00512016082.slib
write_lib ET00512016082
```

The ACE Toolkit Synopsys model output is a directory with MIN, NOM, and MAX lib files. Choose the appropriate file and create a db file with a script similar to the following:

```
read_lib ET00512016082_NOM.lib
write_lib ET00512016082_NOM.db
```

Once the Synopsys models are assembled, you must create the Synopsys initialization file .synopsys_dc.setup in your working directory. See Example 6–9 for an example TSC4000 initialization file including the necessary Synopsys models for our example design.

Example 6–9. Synopsys .synopsys_dc.setup Initialization File for TSC4000

```
company = ”Your Company” ;
designer = ”Your Name”;
view_background = ”black”;

search_path = ”. /home/synopsys_3.5a/libraries/syn /home/
asiclib/tidss_4.2/tsc4000_1.0/sun5/synopsys/lib”;

link_library = ”* TSC4000_3.3V_TLM_NOM_CORE.db
TSC4000_3.3V_TLM_NOM_IO.db TSC4000_3.3V_TC_SOFTMAC.db
EJ0819203208A_NOM.db ET00512016082_NOM.db
T320C2700B0_3.3V_NOM.db”;

target_library = ”TSC4000_3.3V_TLM_NOM_CORE.db
TSC4000_3.3V_TLM_NOM_IO.db EJ0819203208A_NOM.db
ET00512016082_NOM.db T320C2700B0_3.3V_NOM.db”;

symbol_library = ”TSC4000_3.3V_CORE.sdb TSC4000_3.3V_IO.sdb
EJ0819203208A.sdb ET00512016082.sdb T320C2700B0.sdb”;
```

Every other component in the example design is in synthesizable RTL format. Example 6–10 shows a Design Analyzer timing-critical script that can be run to synthesize every component of the design. $DESIGN denotes where the RTL design is located. Every component is saved in db and VHDL format with all of its subcomponents included in the file because of the –hierarchy switch. If only the top-level design file is desired, the compile command can be executed once at the end of the script when all the components have been read into Design Analyzer. This script is a basic template and is not fully optimized. Design specific constraints for synthesis include:

- Clock definition, frequency, and skew
- Drive strength on inputs
- Input delay time from clock
- Load on outputs
- Output delay from clock

Example 6–10. Timing-Critical Design Analyzer Synthesis Script

```
read –format vhdl { $DESIGN/xclock.ent, $DESIGN/xclock.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”xclock.vhd”
write –format db –hierarchy –output ”xclock.db”

read –format vhdl { $DESIGN/xcntrl.ent, $DESIGN/xcntrl.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”xcntrl.vhd”
write –format db –hierarchy –output ”xcntrl.db”

read –format vhdl { $DESIGN/xdecoder.ent, $DESIGN/xdecoder.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”xdecoder.vhd”
write –format db –hierarchy –output ”xdecoder.db”

read –format vhdl { $DESIGN/xemureg.ent, $DESIGN/xemureg.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”xemureg.vhd”
write –format db –hierarchy –output ”xemureg.db”

analyze –format vhdl –lib WORK { $DESIGN/xfifo.ent, $DESIGN/xfifo.rtl}
elaborate xfifo –arch ”rtl” –lib WORK –update –param ”M = 62”
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”xfifo.vhd”
write –format db –hierarchy –output ”xfifo.db”
```

Example 6–10. Timing-Critical Design Analyzer Synthesis Script (Continued)

```
read –format vhdl { $DESIGN/xintfil.ent, $DESIGN/xintfil.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”xintfil.vhd”
write –format db –hierarchy –output ”xintfil.db”

read –format vhdl { $DESIGN/xintsync.ent, $DESIGN/xintsync.rtl}
set_min_fault_coverage 95 –timing_critical
uniquify
compile –map_effort high
write –format vhdl –hierarchy –output ”xintsync.vhd”
write –format db –hierarchy –output ”xintsync.db”

read –format vhdl { $DESIGN/xlatch.ent, $DESIGN/xlatch.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”xlatch.vhd”
write –format db –hierarchy –output ”xlatch.db”

read –format vhdl { $DESIGN/xperreg.ent, $DESIGN/xperreg.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”xperreg.vhd”
write –format db –hierarchy –output ”xperreg.db”

read –format vhdl { $DESIGN/xrwblk.ent, $DESIGN/xrwblk.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”xrwblk.vhd”
write –format db –hierarchy –output ”xrwblk.db”

read –format vhdl { $DESIGN/xwgen.ent, $DESIGN/xwgen.rtl}
set_min_fault_coverage 95 –timing_critical
uniquify
compile –map_effort high
write –format vhdl –hierarchy –output ”xwgen.vhd”

write –format db –hierarchy –output ”xwgen.db”
read –format vhdl { $DESIGN/xintf.ent, $DESIGN/xintf.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”xintf.vhd”
write –format db –hierarchy –output ”xintf.db”

read –format vhdl $DESIGN/xmem_glue_logic.vhd
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”xmem_glue_logic.vhd”
write –format db –hierarchy –output ”xmem_glue_logic.db”
```

Example 6–10. Timing-Critical Design Analyzer Synthesis Script (Continued)

```
read –format vhdl { $DESIGN/cromw.ent, $DESIGN/cromw.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”cromw.vhd”
write –format db –hierarchy –output ”cromw.db”

read –format vhdl { $DESIGN/sramwagen.ent, $DESIGN/sramwagen.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”sramwagen.vhd”
write –format db –hierarchy –output ”sramwagen.db”

read –format vhdl { $DESIGN/sramwdec.ent, $DESIGN/sramwdec.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”sramwdec.vhd”
write –format db –hierarchy –output ”sramwdec.db”

read –format vhdl { $DESIGN/sramwfsm.ent, $DESIGN/sramwfsm.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”sramwfsm.vhd”
write –format db –hierarchy –output ”sramwfsm.db”

read –format vhdl { $DESIGN/cromw.ent, $DESIGN/cromw.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”cromw.vhd”
write –format db –hierarchy –output ”cromw.db”

read –format vhdl { $DESIGN/sramwagen.ent, $DESIGN/sramwagen.rtl}
set_min_fault_coverage 95 –timing_critical
compile –map_effort high
write –format vhdl –hierarchy –output ”sramwagen.vhd”
write –format db –hierarchy –output ”sramwagen.db”
```

6.8 Simulating the Gate-Level Synthesis Output

When synthesis is complete, you have a library-specific, gate-level netlist for your design. This netlist is a better model for timing, and should be simulated to ensure that you did not acquire any timing errors with the implementation of your library. You must add the following library clauses before every entity in your synthesis output to properly load the library components into the simulator:

```
library tsc4000;
use tsc4000.all;
```

The following line should be added to the Library section of quickhdl.ini to map library TSC4000 to the proper TIDSS location:

```
tsc4000 = /home/tidss_4.2/tsc4000_1.0/sun5/quickhdl/lib/3.3v
```

See Example 6–11 for an updated QuickHDL compile script.

Example 6–11. Updated Gate-Level QuickHDL Compile Script

```
### Compile ROM program file
qvhcom –explicit –work TI_CUSTOM \
EJ0819203208A_prog.vhd

### Compile memories
qvhcom –explicit \
ET00512016082.vhd \
ET02048016042.vhd \
EJ0819203208A.vhd

### Compile T320C2700B0 core
qvhcom \
ti_bus_timing.vhd \
watchblock.vhd \
SC_ANKOOR_C3.vhd \
T320C2700func.vhd \
T320C2700B0.vhd

### Compile Design Analyzer output and testbench
qvhcom example_design.vhd \
test_example_design.vhd
```

Chapter 7

Electrical Considerations

To ensure that the core performs in accordance with the rated specifications, you need to account for certain electrical considerations in your design.

This chapter describes the minimum electrical requirements for ensuring core performance.

Topic **Page**

7.1 Minimum Operating Voltage for the cDSP Chip

To arrive at the minimum operating voltage (V_{DDmin}) for the cDSP chip you must consider the following:

- ❑ Minimum voltage at which the core is characterized (V_{min_char})
- ❑ Voltage drop in the power network of the core (DeltaVcore)
- ❑ Voltage drop in the power network of the chip (DeltaVchip)
- ❑ Voltage drop in the bond wires and package leads (DeltaVpack)

V_{DDmin} is the sum of V_{min_char}, DeltaVcore, DeltaVchip, and DeltaVpack.

Based on initial results for the T320C2700B0 core, V_{min_char} equals 1.65 volts and DeltaVcore equals 60 millivolts (mV) tentatively.

DeltaVchip and DeltaVpack are specific to the cDSP chip design and the package used. To calculate these values, use a preliminary value of 200 milliamperes (mA) for power dissipation of the core.

Using this minimum VDD specification at the chip/board level ensures that the cell macros within the core see at least the minimum voltage for which they are characterized. This, in turn, ensures the rated core performance. If these minimum specifications are not met, there may be degradation in the performance of the core in your cDSP design.

To maximize core performance at the chip level, you must also consider the power network and the package/pin selection in your cDSP design. For more information, see the *TMX320C2700B0-E3 data sheet* and the *TSC6000 0.18 µm CMOS Design Manual*.

7.2 Clock Considerations

Since there is exchange of data between the three clock domains, the three independent clocks, CPUCLK, IMUCLK, and SYSCLK must be balanced within a reasonable skew for the chip to function properly. The CPUCLK domain starts from the CPUCLKOUT pin of the core and goes inside the core again through the CPUCLKIN pin after passing through a chip-level clock-tree synthesis (CTS) buffer. Similarly, the IMUCLK domain starts from IMUCLKOUT pin of the core and goes inside the core again through IMUCLKIN pin after passing through a chip-level CTS buffer. The SYSCLK domain is a chip-level clock domain that starts from the SYSCLKOUT pin of the core and is used for chip-level clocking. In a cDSP using the T320C2700B0 core, the skew across the three clock domains is balanced by ensuring that all three see the same insertion delay.

Within the core, CPUCLK and IMUCLK are matched; therefore, they have the same insertion delay. The core insertion delay for CPUCLKIN and IMUCLKIN equals 1.203 ns at IND_MAX condition.

Figure 7–1 shows the clocking scheme for the T320C2700B0 core.

The value of the insertion delay to be specified for each CTS buffer is determined as follows:

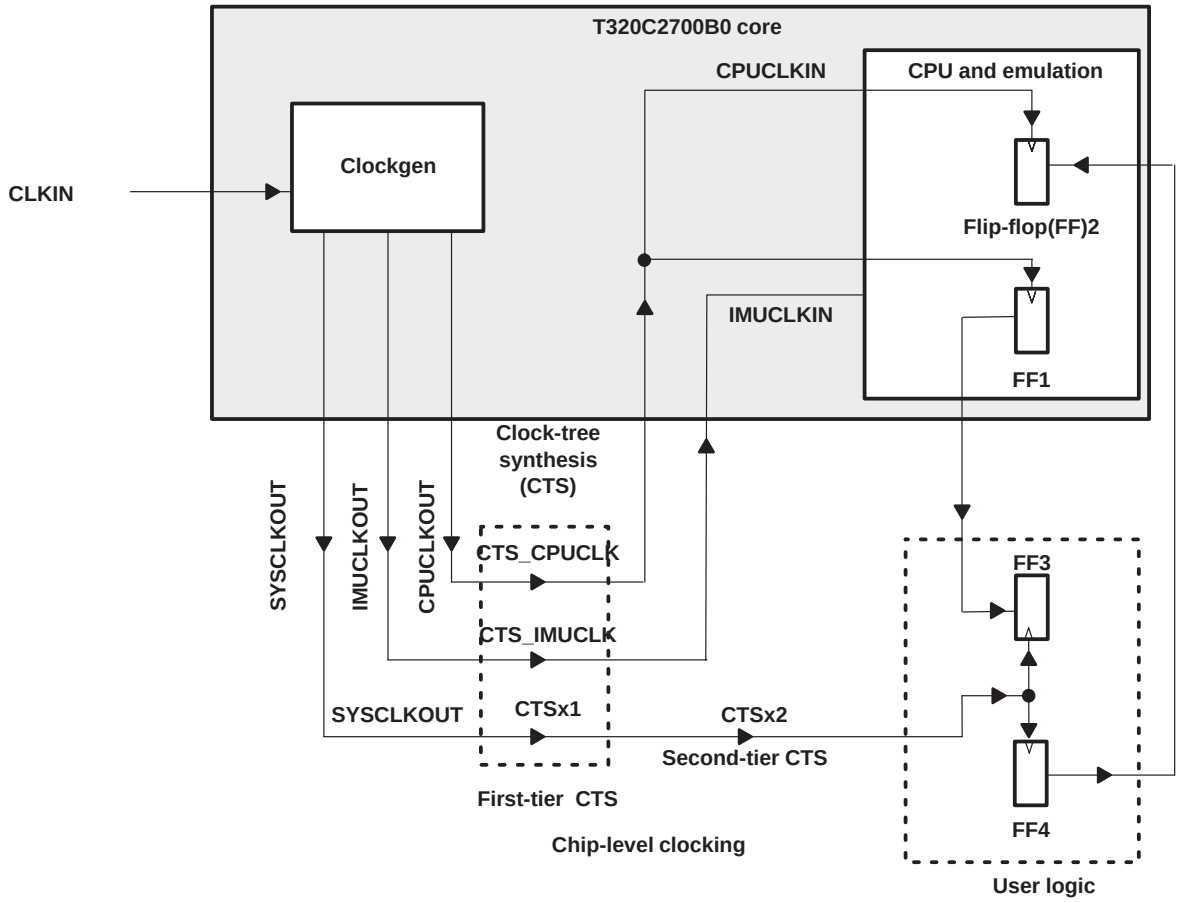
- Tins_CORE equals insertion delay of the clock tree within the core.
- Tins_CTSx1 equals insertion delay of the CTS macro used for first-tier clocking for the user logic.
- Tins_CTSx2 equals insertion delay of the CTS macro used for second-tier clocking for the user logic.
- Tins_CPUCLK equals insertion delay that must be specified for the chip-level CTS buffer for CPUCLK (CTS_CPUCLK macro).
- Tins_IMUCLK equals insertion delay that must be specified for the chip-level CTS buffer for IMUCLK (CTS_IMUCLK macro).

For all the clocks to be balanced, the insertion delays must be specified such that the following condition holds true:

$$Tins\_CORE + Tins\_CPUCLK = Tins\_CTSx1 + Tins\_CTSx2$$

Note that CTS_CPUCLK, CTS_IMUCLK, and CTSx1 must be the same CTS macro, so that the three clock outputs from the core see the same loading. Since the insertion delay for the CPUCLK and IMUCLK inside the core is the same, Tins_IMUCLK equals Tins_CPUCLK at the chip level.

Figure 7–1. Clocking Scheme for the T320C2700B0 Core



A second level of fine tuning is needed if there is any skew between the arrival of CPUCLKOUT, IMUCLKOUT, and SYSCLKOUT to the root of the respective CTS buffers in the chip. This skew must be extracted after the layout and must be fed to the CTS flow for a second level of fine tuning. If the CTS buffers at the chip level are placed close to the core clock outputs, there is negligible difference between the arrival time of the three clocks to the root of the CTS buffers. In this way, a second level of fine tuning can be avoided.

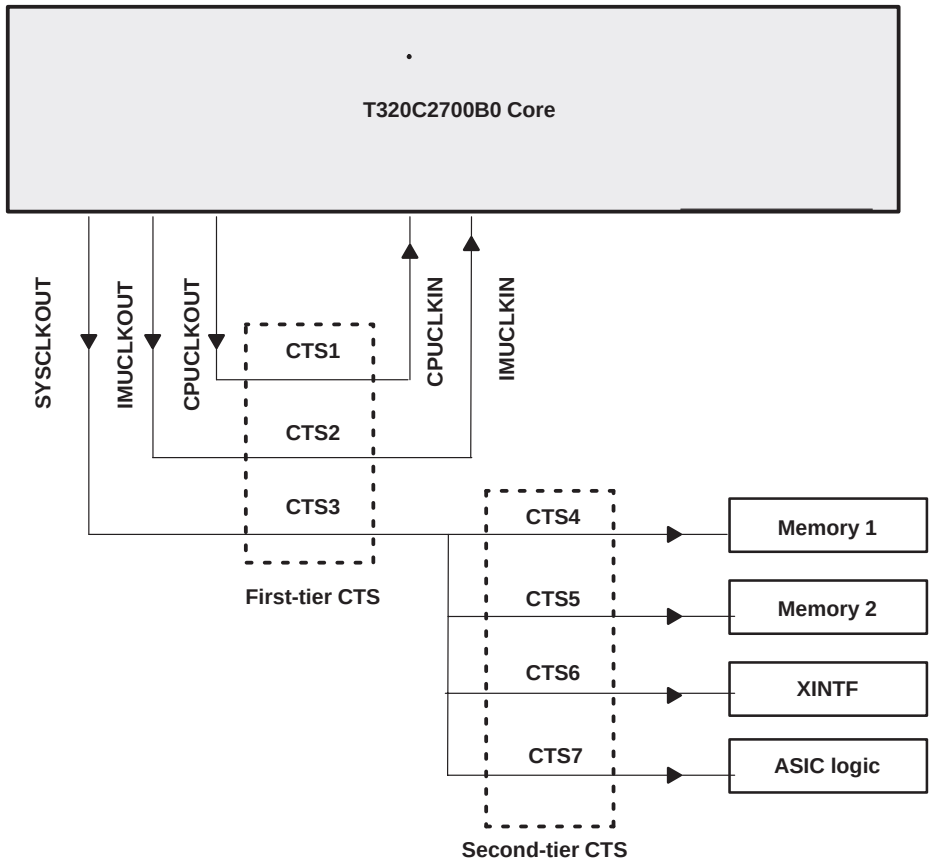
If Tins_CTSxx is greater than Tins_CORECPUCLK, the CPUCLK and IMUCLK are brought out of the core instead of being connected internally to the CPU. In this case, addition of an insertion delay in the CPUCLK and IMUCLK domains is required. This delay can be done only at the chip level because the core is a fixed module.

See the *TSC6000 0.18-µm CMOS Standard Cell Macro Library Summary* for more information on the clocking scheme.

7.3 Example of Chip-Level Clocking

Consider a design that has two separate memory modules with their respective wrappers, an external interface block, and a separate group of ASIC logic. There could be several ways to plan the clocking of this design. Figure 7–2 shows one way of doing it. The important thing to note here is the reasoning behind this scheme and the method used to assign insertion delays for the CTS buffers.

Figure 7–2. Example of Chip-Level Clocking Using the T320C2700B0 Core



The four different chip-level clock domains are fed from separate CTS buffers so that the problem of clock balancing in each domain remains manageable. In TI ASIC CTS flow, the clock balancing is done from the root of the CTS buffer to the clock inputs of the target flip-flops.

In order to make sure that the delays from IMUCLKOUT, CPUCLKOUT, and SYSCLKOUT to the root of respective CTS buffers are matched, the three

CTS macros must be placed very close to the three core clock outputs. If the four chip-level CTS buffers are connected directly to SYSCLKOUT, then SYSCLKOUT sees a different loading than CPUCLKOUT and IMUCLKOUT, which are connected to just one CTS buffer. This causes the delay from SYSCLKOUT pin to the input of cts3 to be different than the delay from CPUCLKOUT/IMUCLKOUT to the input of cts1/cts2. Hence, this scheme has a two-tier clocking at the chip level. First, SYSCLKOUT goes to cts3 and cts3 then gets connected to all the CTS buffers (cts4-7), which drive the chip-level clock domains.

The insertion delay specification for the different CTS buffers is as follows:

- Tins_core equals insertion delay of the clocking inside the core.
- Tins_core equals 1.203ns for the T320C2700B0 core at IND_MAX conditions.
- Tins_cts1_2 equals insertion delay to be specified for buffers cts1 and cts2. This insertion delay must be the same as those for cts1 and cts2.
- Tins_cts3 equals insertion delay to be specified for buffer cts3.
- Tins_cts4_7 equals insertion delay to be specified for buffers cts4, cts5, cts6, and cts7.

Therefore, Tins_cts1_2, Tins_cts3, and Tins_cts4_7 must be chosen in such a way that the following condition holds true:

$$\text{Tins\_core} + \text{Tins\_cts1\_2} = \text{Tins\_cts3} + \text{Tins\_cts4\_7}$$

In addition, you must make the CTS macros cts1, cts2, and cts3 a user group and place it adjacent to the clock outputs, CPUCLKOUT, IMUCLKOUT, and SYSCLKOUT. This will ensure that the wires connecting the clock outputs to the respective CTS macros are matched.

Once the post layout CTS report is received, you must ensure that the insertion delay that is achieved (apart from the skew) is the same as the one you specified. This is extremely important because any difference in the insertion delay will add directly to the skew across various domains.

7.4 Frequently Asked Questions About T320C2700B0 Clocking

Question: Why is there a need for the CTS on core inputs, CPUCLKIN and IMUCLKIN, as specified in the E3 data sheet? Are these two clocks not already balanced inside the core?

Answer: For the chip to work properly, the CPUCLK, IMUCLK, and SYSCLK must be balanced within a skew limit between each other. To do this, you must match the total insertion delay in all three domains. Since the insertion delay on SYSCLK could be higher for larger chips than for IMUCLK and CPUCLK inside the core, you need the flexibility to add an appropriate insertion delay to both CPUCLK and IMUCLK. The CTS buffers bring out the CPUCLKOUT and IMUCLKOUT and connect them back to CPUCLKIN and IMUCLKIN.

The CPUCLKIN and IMUCLKIN are balanced inside the core.

Question: Why do we have clock balancing requirements between the core, XINTF, memory wrappers, and the ASIC logic? How can we balance the clocks between these domains?

Answer: In general, clocks between any two synchronous domains need to be balanced if they exchange data. Data is transferred between the core and the wrappers, as well as between the core and the XINTF. Hence, the clocks going to these domains must be balanced. If you take care of balancing the skew across CPUCLK, IMUCLK, and SYSCLK the clocks will automatically balance.

Question: In single-phase clocking schemes, how can we make the design less sensitive to clock skews?

Answer: In single-phase clocking schemes, you must be very careful of hold-time problems. This is because hold-time violations are related to clock edges and not to clock period. Therefore, if your design has hold-time violations, it will continue to have problems regardless of how slow the clock is run. The problem with measuring clock skew is that it depends on many factors such as the process gradient across the die and thermal gradients within the chip. It is difficult to comprehend such variations during clock-skew analysis. To fix any hold-time violations, you must keep some margin in the design.

One way to make the design less prone to hold-time problems resulting from high clock skew is to add delay elements to fast paths like the scan chain. The Synopsys Design Compiler allows for automatic fixing of hold time through use of the ‘fix_hold_time’ command. Use a pessimistic clock skew and an optimistic wire load model before doing ‘fix_hold_time’ in the Synopsys Design Compiler.

Appendix A

Signal Descriptions (TSC6000 ASIC Library)

This chapter describes the ’C2700B0 signals, which include the signals that are available for use with the customer-defined logic and signals that are used for integrated memories interface.

Topic **Page**

A.1 T320C2700B0 Core Signals

The ’C2700B0 core includes the following five groups of signals:

- Memory-interface signals (see Table A–1 on page A-4)
- Control and status signals (see Table A–2 on page A-11)
- Write/read protection mode signals (see Table A–3 on page A-14)
- Reset and interrupt signals (see Table A–4 on page A-15)
- Emulation signals (see Table A–5 on page A-17)
- Visibility port signals (see Table A–6 on page A-19)

A.1.1 Memory Interface Signals

All memory-interface input and output signals (Figure A–1) are used by the core to read and write memories. The program-read data bus (PRDY) signals and the data-read data bus (DRDY) signals convey information which indicates that the memory interface signals are sending back read data to a read request. Any memory or interface bridge that drives data onto the program-read or data-read bus also must assert the corresponding PRDY or DRDY to indicate that the data bus is being driven.

The memory-interface signals are listed in Table A–1 beginning on page A-4.

Figure A–1. Memory-Interface Signals Diagram

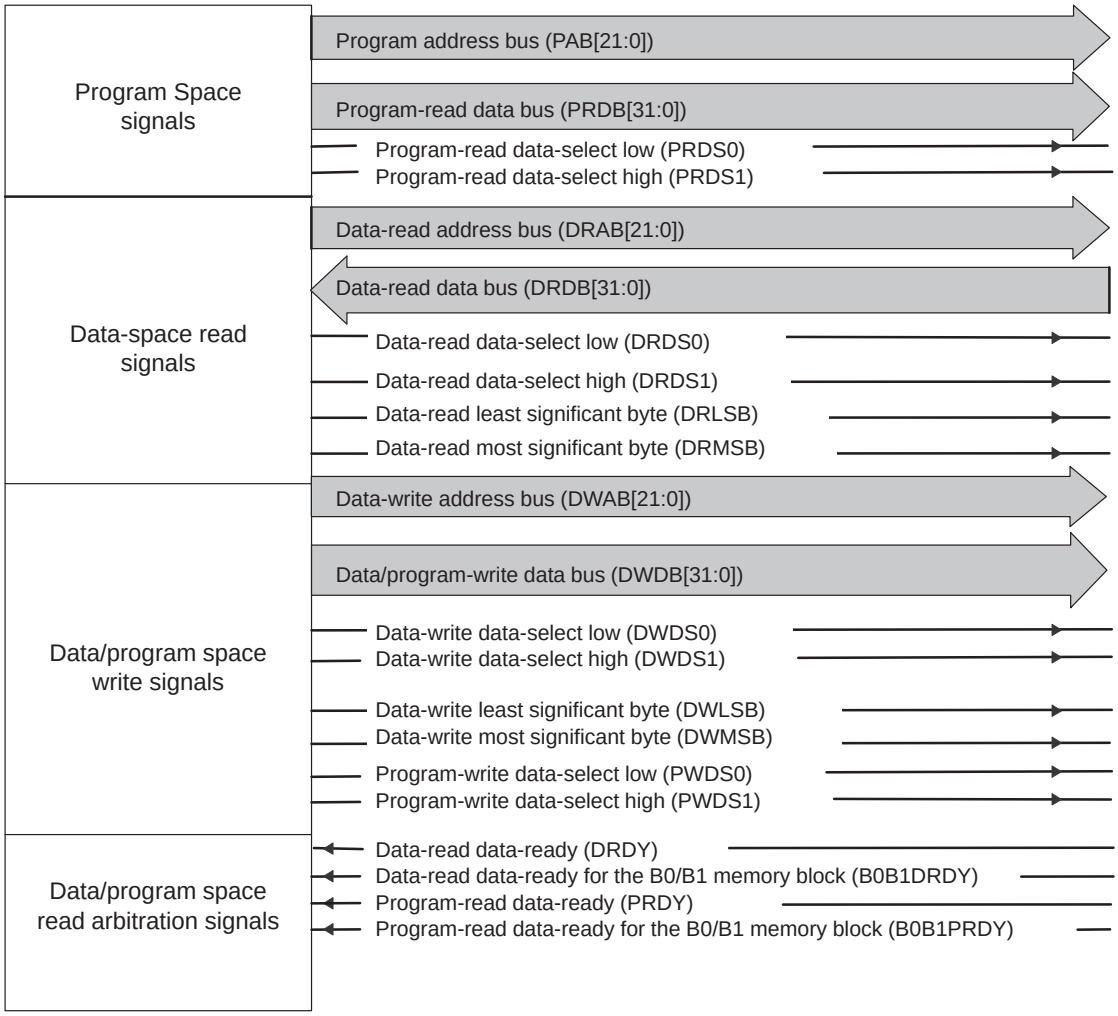


Table A–1. Memory Interface Signal Descriptions

(a) Program-Space Signals

Signal Name	I/O†	Description
PAB[21:0]	O	Program address bus. PAB is a 22-bit buses that provides the address for both reads and writes to program space. PAB[0] is the LSB. PAB[21:0] go active on the rising edge of SYSCLKOUT and remains active until the next memory cycle starts.
PRDB[31:0]‡	I	Program-read data bus. PRDB[31:0] is a 32-bit program-space bus, defined for operations assuming banked memory (i.e., memories with one 16-bit bank for odd addresses and another 16-bit bank for even addresses). The odd bank always drives the higher half of the bus, while the even bank always drives the lower half of the bus. Both banks are enabled by 32-bit accesses. Data is always latched on the rising edge of SYSCLKOUT at the end of a program read cycle.
PRDS0 PRDS1‡	O	Program-read data-select low and program-read data-select high. PRDS0, when active (high), indicates that the CPU is requesting a 16-bit read from an even-address program-space location. PRDS1, when active (high), indicates that the CPU is requesting a 16-bit read from an odd-address program-space location. PRDS0 and PRDS1 go active on the rising edge of SYSCLKOUT and remain active until the next memory cycle starts. PRDS0 and PRDS1 are used to decode the following bus transactions:

PRDS1	PRDB[31:16]	PRDS0	PRDB[15:0]	Transaction Type
High	Data(16)	High	Data(16)	32-bit read, even aligned
Low	Undefined	High	Data(16)	16-bit read, even address
High	Data(16)	Low	Undefined	16-bit read, odd address
Low	Undefined	Low	Undefined	No transaction

(b) Data-Space Read Signals

Signal Name	I/O†	Description
DRAB[21:0]	O	Data-read address bus. DRAB is a 22-bit bus that outputs the data-read addresses from the CPU. DRAB[0] is the least significant bit (LSB). DRAB goes active on the rising edge of SYSCLKOUT and remains active until the next memory cycle starts.
DRDB[31:0]	I	Data-read data bus. DRDB[31:0] is a 32-bit bus that inputs 16- or 32-bit data-read data. The data is latched into the core on the rising edge of SYSCLKOUT at the end of the data-read cycle.

† I = Input, O = Output

‡ Instructions are normally fetched as 32-bit program read operations, except at the start of a discontinuity. If the address starts at an odd location, only PRDS1 is driven high and the instruction is fetched on PRDB[31:16]. The core ignores any data present on PRDB[15:0]

Table A–1. Memory Interface Signal Descriptions (Continued)

(b) Data-Space Read Signals (Continued)

Signal Name	I/O†	Description
DRDS0 DRDS1	O	Data-read data select low and data-read data-select high. DRDS0, when active (high), indicates that the CPU is requesting a 16-bit read from an even-address data-space location. DRDS1, when active (high), indicates that the CPU is requesting a 16-bit read from an odd-address data-space location. DRDS0 and DRDS1 go active at the rising edge of SYSCLKOUT and remain active until the next memory cycle starts. DRDS0 and DRDS1 are used to decode the following bus transactions:

DRDS1	DRDB(31:16)	DRDS0	DRDB(15:0)	Transaction Type
High	Data(16)	High	Data(16)	32-bit read, even aligned
Low	Undefined	High	Data(16)	16-bit read, even address
High	Data(16)	Low	Undefined	16-bit read, odd address
Low	Undefined	Low	Undefined	No transaction

DRLSB DRMSB	O	Data-read least significant byte (LSByte) and data-read most significant byte (MSByte). DRLSB, when active (high), indicates that the CPU is performing a 16-bit read operation with the LSByte of the word being the active data element. DRMSB, when active (high), indicates that the CPU is performing a 16-bit read operation with the MSByte of the word as the active data element. DRLSB and DRMSB go active at the rising edge of SYSCLKOUT and remain active until the next memory cycle starts. The DRLSB and DRMSB signals can be used in conjunction with the DRSD0 and DRSD1 signals to perform byte-only accesses from byte-wide peripherals or memory. This usage can improve cycle performance on accesses to such peripherals. The decode of the signals is as follows:

DRDS1	DRDS0	DRMSB	DRLSB	Transaction Type
High	High	High	High	32-bit read
Low	High	High	High	16-bit read, even address
Low	High	Low	High	Read, LSByte
Low	High	High	Low	Read, MSByte
High	Low	High	High	16-bit read, odd address
High	Low	Low	High	Read, LSByte

† I = Input, O = Output

‡ Instructions are normally fetched as 32-bit program read operations, except at the start of a discontinuity. If the address starts at an odd location, only PRDS1 is driven high and the instruction is fetched on PRDB[31:16]. The core ignores any data present on PRDB[15:0]

Table A–1. Memory Interface Signal Descriptions (Continued)

(b) Data-Space Read Signals (Continued)

Signal Name	I/O†	Description				
		High	Low	High	Low	Read, MSByte
		Low	Low	Low	Low	No transaction

† I = Input, O = Output

‡ Instructions are normally fetched as 32-bit program read operations, except at the start of a discontinuity. If the address starts at an odd location, only PRDS1 is driven high and the instruction is fetched on PRDB[31:16]. The core ignores any data present on PRDB[15:0]

Table A–1. Memory Interface Signal Descriptions (Continued)

(c) Data-Space Write Signals

Signal Name	I/O†	Description
DWAB [21:0]	O	Data-write address bus. DWAB is a 22-bit buses that outputs the data-write addresses from the CPU. DWAB is the LSB. DWAB goes active on the rising edge of SYSCLKOUT and remains active until the next memory cycle starts.
DWDB [31:0]	O	Data/program-write data bus. DWDB is a 32-bit buses that outputs 16- or 32-bit data to data or program space. This data is written out on the rising edge of SYSCLKOUT and remains active until the end of the data-write cycle.
DWDS0 DWDS1	O	Data-write data-select low and data-write data-select high. DWDS0, when active (high), indicates that the CPU is requesting a 16-bit write to an even-address data-space location. DWDS1, when active (high), indicates that the CPU is requesting a 16-bit write to an odd-address data-space location. DWDS0 and DWDS1 go active at the rising edge of SYSCLKOUT and remain active until the next memory cycle starts. DWDS0 and DWDS1 are used to decode the following bus transactions:

DWDS1	DWDB(31:16)	DWDS0	DWDB(15:0)	Transaction Type
High	Data(16)	High	Data(16)	32-bit write, even aligned
Low	Undefined	High	Data(16)	16-bit write, even address
High	Data(16)	Low	Undefined	16-bit write, odd address
Low	Undefined	Low	Undefined	No transaction

† I = Input, O = Output

‡ Instructions are normally fetched as 32-bit program read operations, except at the start of a discontinuity. If the address starts at an odd location, only PRDS1 is driven high and the instruction is fetched on PRDB[31:16]. The core ignores any data present on PRDB[15:0]

Table A–1. Memory Interface Signal Descriptions (Continued)

(c) Data-Space Write Signals (Continued)

Signal Name	I/O†	Description
DWLSB DWMSB	O	Data-write LSByte, data-write MSByte. DWLSB, when active (high), indicates that the CPU is performing a 16-bit write operation with the LSByte of the word being the active data element. DWMSB, when active (high), indicates that the CPU is performing a 16-bit write operation with the MSByte of the word being the active data element. DWLSB and DWMSB go active at the rising edge of SYSCLKOUT and remain active until the next memory cycle starts. The DWLSB and DWMSB signals can be used with the DWDS0 and DWDS1 signals to perform byte-only accesses from byte-wide peripherals or memory to improve cycle performance on accesses to such peripherals. The decode of the signals is as follows:

DWDS1	DWDS0	DWMSB	DWLSB	Transaction Type
High	High	High	High	32-bit write, even aligned
Low	High	High	High	16-bit write, even address
Low	High	Low	High	Write, LSByte
Low	High	High	Low	Write, MSByte
High	Low	High	High	16-bit write, odd address
High	Low	Low	High	Write, LSByte
High	Low	High	Low	Write, MSByte
Low	Low	Low	Low	No transaction

Signal Name	I/O†	Description
PWDS0 PWDS1	O	Program-write data-select low and program-write data-select high. PWDS0, when active (high), indicates that the CPU is requesting a 16-bit write to an even-address program-space location. PWDS1, when active (high), indicates that the CPU is requesting a 16-bit write to an odd-address program-space location. PWDS0 and PWDS1 go active at the rising edge of SYSCLKOUT and remain active until the next memory cycle starts (determined by POREADY signals to the core). PWDS0 and PWDS1 are used to decode the following bus transactions:

PWDS1	PWDB(31:16)	PWDS0	PWDB(15:0)	Transaction Type
High	Data(16)	High	Data(16)	32-bit write, even aligned
Low	Undefined	High	Data(16)	16-bit write, even address

† I = Input, O = Output

‡ Instructions are normally fetched as 32-bit program read operations, except at the start of a discontinuity. If the address starts at an odd location, only PRDS1 is driven high and the instruction is fetched on PRDB[31:16]. The core ignores any data present on PRDB[15:0]

Table A–1. Memory Interface Signal Descriptions (Continued)

(c) Data-Space Write Signals (Continued)

Signal Name	I/O†	Description				
		High	Data(16)	Low	Undefined	16-bit write, odd address
		Low	Undefined	Low	Undefined	No transaction

† I = Input, O = Output

‡ Instructions are normally fetched as 32-bit program read operations, except at the start of a discontinuity. If the address starts at an odd location, only PRDS1 is driven high and the instruction is fetched on PRDB[31:16]. The core ignores any data present on PRDB[15:0]

Table A–1. Memory Interface Signal Descriptions (Continued)

(d) Program/Data-Space Read Arbitration Signals

Signal Name	I/O†	Description
DRDY	I	Data-read data ready. DRDY, when active (high), indicates that the data on the DRDB[31:0] bus is available to the core. DRDY is daisy-chained through memory blocks. The input to the first block of DRDY must be tied low. If the application includes no data space blocks, DRDY must be tied low at the input to the port. Blocks B0 and B1 must be daisy-chained through the B0B1DRDY signal and not daisy-chained through the DRDY signal (to which all other memory blocks or peripherals are connected). This is required for isolation during core functional tests.
B0B1DRDY	I	Data-read data ready for the B0 memory block. B0B1DRDY is the same as the DRDY, except that the signal is connected only to the B0/B1 memory blocks. B0B1DRDY is asserted by the B0/B1 block to indicate to the CPU that the DRDB is being driven with read data in response to a data-read request.
PRDY	I	Program-read data ready. PRDY, when active (high), indicates that the data on the PRDB[31:0] bus is available to the core. This signal is daisy-chained through memory blocks or peripherals. The input to the first block of PRDY must be tied low. If the application contains no program space blocks, PRDY must be tied low at the input to the port. Blocks B0 and B1 must be daisy-chained through the B0B1PRDY signal and not daisy-chained through the PRDY signal (to which all other memory blocks are connected). This is required for isolation during core functional tests.
B0B1PRDY/	I	B0B1PRDY is similar to the PRDY but is connected to the B0/B1 block and cannot be connected to anything else. B0B1PRDY is asserted by the B0/B1 block to indicate to the CPU that the PRDB is being driven with read data in response to a program-read request.

† I = Input, O = Output

‡ Instructions are normally fetched as 32-bit program read operations, except at the start of a discontinuity. If the address starts at an odd location, only PRDS1 is driven high and the instruction is fetched on PRDB[31:16]. The core ignores any data present on PRDB[15:0]

A.1.2 Control and Status Signals

The control and status signals (Figure A–2) are listed in Table A–2, page A-11).

Figure A–2. Control and Status Signals Diagram

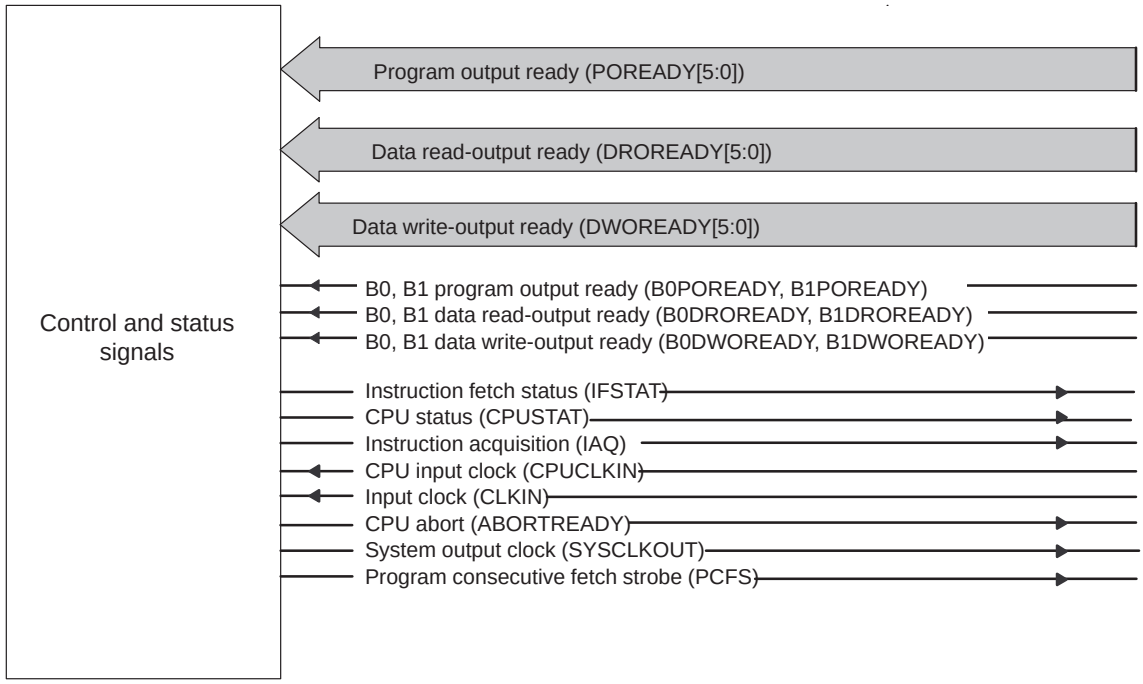


Table A–2. Control and Status Signal Descriptions

Signal Name	I/O†	Description
ABORTREADY	O	CPU aborting all requests. ABORTREADY is an output signal that must be connected to all memory interface components. This signal, when active (high), indicates that the CPU is aborting all the existing requests.
B0DROREADY	I	Data-space memory-read output ready for connection to the B0 memory block. B0DROREADY is identical to the DROREADY signal except that this is for connection to the B0 block. No other memory can be connected to B0DROREADY.
B0DWOREADY	I	Data-space memory-write output ready for the B0 memory block. B0DWOREADY is identical to the DWOREADY signal except that this is for connection to the B0 memory block. No other memory must connect to this signal.

† I = Input, O = Output.

Table A–2. Control and Status Signal Descriptions (Continued)

Signal Name	I/O†	Description
B0POREADY	I	Program-space-memory output ready for the B0 memory block. B0POREADY is identical to the POREADY signal except that B0POREADY is meant for connection to the B0 block. No other memory block can be connected to this signal.
B1DROREADY	I	Data-space memory-read output ready for connection to the B1 memory block. B1DROREADY is identical to the DROREADY signal except that this is for connection to the B1 block. No other memory can be connected to B1DROREADY.
B1DWOREADY	I	Data-space memory-write output ready for the B1 memory block. B1DWOREADY is identical to the DWOREADY signal except that this is for connection to the B1 memory block. No other memory must connect to this signal.
B1POREADY	I	Program-space-memory output ready for the B1 memory block. B1POREADY is identical to the POREADY signal except that B1POREADY is meant for connection to the B1 block. No other memory block can be connected to this signal and B1 can be mapped only to data memory. B1POREADY must be tied high.
CLKIN	I	Input clock. CLKIN is the input-clock feed to the device core. The input frequency is proportional to the CPU cycle time. For example, for a device operating at CLKIN = 100 MHz, CPU cycle time is 10 ns.
CPUCLKIN	I	CPU input clock signal. The output CPUCLKOUT should be connected through a clock-tree synthesis (CTS) macro to CPUCLKIN.
CPUCLKOUT	O	CPU output clock. CPUCLKOUT is generated by the CPU. For details on connecting CPUCLKOUT, see the clocking section.
CPUSTAT	O	CPU status. CPUSTAT, when active (high), indicates that the CPU is ready to start the memory cycles requested for the memory interface. If CPUSTAT is inactive (low), the CPU is in a CPU wait stated cycle and cannot accept data.
DROREADY[5:0]	I	Data-space memory-read output ready. The six DROREADY signals are used by memories and interface bridges to request waitstates on data-read operation. When a memory or interface bridge cannot complete the requested data-read access, it pulls the DROREADY[5:0] signals low. Unused DROREADY signals must be tied high. The memory read operation is two-staged and the CPU expects data to be driven one cycle after it samples the DROREADY signal to be high. *This relationship is very important, and memories and interface bridges must adhere to this. If not, the CPU can produce erroneous results.*

† I = Input, O = Output.

Table A–2. Control and Status Signal Descriptions (Continued)

Signal Name	I/O†	Description
DWOREADY[5:0]	I	Data-space memory-write output ready. The six DWOREADY signals are connected to the CPU. The devices that have a pending data-write operation must generate a ready (high) or not-ready (low) condition on their respective DWOREADY signals to keep the CPU from completing the machine cycle until all scheduled memory operations are complete. All unused DWOREADY signals must be tied high. A not-ready condition on the DWOREADY line does not necessarily stall the fetch mechanism.
IAQ	O	Instruction acquisition. IAQ, when active (high), indicates that the current bus cycle is performing an instruction fetch. If IAQ is low and a program-space operation is in progress, a program-space data-read or -write operation is being performed. This signal is used to distinguish between an instruction fetch and a program read or write operation (performed by the PREAD, PWRITE, and MAC instructions). IAQ is valid on the rising edge of SYSCLKOUT and remains valid until the next memory cycle starts (determined by POREADY signals to the core).
IFSTAT	O	Instruction-fetch status. IFSTAT, when active (high), indicates that the instruction-fetch mechanism is ready to start the memory cycles requested for the memory interface. If IFSTAT is inactive (low), the instruction-fetch mechanism is in a program-space waitstated cycle.
IMUCLKIN	I	Emulation domain input-clock signal. The output IMUCLKOUT must be connected through a CTS macro to IMUCLKIN.
IMUCLKOUT	O	Emulation domain clock output. For details of connecting IMUCLKOUT, see the CTS section.
POREADY[5:0]	I	Program-space-memory output ready. POREADY signals are used by memories and interface bridges to request wait states on program memory operations. When a memory or interface bridge cannot complete the requested program-space access, POREADY is pulled low. Multiple signals are provided so that each block can have an independent POREADY line to the CPU. Unused POREADY signals must be tied high. A not-ready condition on the POREADY line does not necessarily stall the CPU.
PCFS	O	Program consecutive fetch strobe. PCFS indicates that the current fetch is consecutive to the previous fetch operation. PCFS is valid on the rising edge of SYSCLKOUT and remains valid until the end of the program read cycle.
SYSCLKOUT	O	System output clock. SYSCLKOUT is generated by the ’C2700B0 core. The memory-interface components should connect to SYSCLKOUT.

† I = Input, O = Output.

A.1.3 Write/Read Protection Mode Signals

The write/read protection mode signals are listed in Table A–3.

Table A–3. Write/Read Protection Mode Signal Descriptions

Signal Name	I/O†	Description
ENPROT	I	Enable write followed by read-protection mode. ENPROT, when high, enables the write followed by read protection mode within the memory range specified by the PROTSTART and PROTRANGE inputs. If ENPROT is low, the protection mode is disabled and the PROTRANGE and PROTSTART inputs are ignored. ENPROT must be latched on every cycle.
PROTSTART [15:0]	I	Write followed by read-protection mode start-address inputs. These sixteen input signals specify the protection mode start address. The minimum resolution is 64 words. The start address must be a multiple of the protrange value. For example, if the PROTRANGE value is set for 4K, the start address must be a multiple of 4K. The mapping of the PROTSTART signals to the memory-bus address lines is as follows: **Address lines** A21 A20 A19 .. A8 A7 A6 **PROTSTART signal** 15 14 13 .. 2 1 0 PROTSTART[15:0] must be latched on every cycle.
PROTRANGE [15:0]	I	Data-write data-select low and data-write data-select high. DWDS0, when active (high), indicates that the CPU is requesting a 16-bit write to an even-address data-space location. DWDS1, when active (high), indicates that the CPU is requesting a 16-bit write to an odd-address data-space location. DWDS0 and DWDS1 go active at the rising edge of SYSCLKOUT and remain active until the next memory cycle starts. DWDS0 and DWDS1 are used to decode the following bus transactions:

0	1	2	3	...	12	13	14	15	Range Size
1	1	1	1	...	1	1	1	1	**4M**
1	1	1	1	...	1	1	1	0	**2M**
1	1	1	1	...	1	1	0	0	**1M**
1	1	1	1	...	1	0	0	0	**512K**
.	.	.	.		.	.	.	.	**.**
.	.	.	.		.	.	.	.	**.**
.	.	.	.		.	.	.	.	**.**
1	1	1	0	...	0	0	0	0	**512**
1	1	0	0	...	0	0	0	0	**256**
1	0	0	0	...	0	0	0	0	**128**
0	0	0	0	...	0	0	0	0	**64**

A.1.4 Reset and Interrupt Signals

The reset and interrupt signals (Figure A–3) are listed in Table A–4.

Figure A–3. Reset and Interrupt Signals

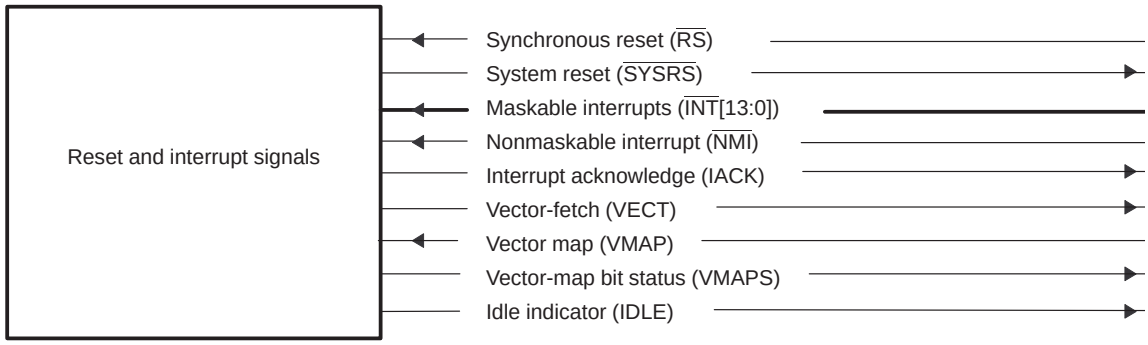


Table A–4. Reset and Interrupt Signal Descriptions

Signal Name	I/O†	Description
IACK	O	Interrupt acknowledge. IACK is driven active (high) by the execution of the IACK 16-bit instruction. The IACK instruction writes the 16-bit immediate value to the data-write bus, DWDB[15:0]. The data-write-address bus, DWAB[21:0], contains the address of the last operation on the bus prior to the IACK instruction and has no relationship to the IACK operation. This data is written out on the rising edge of SYSCLKOUT and remains active until the end of the data-write cycle.
IDLE	O	Idle indicator. IDLE, when active (high), indicates that the CPU has executed an IDLE instruction. The IDLE signal can be used to turn off customer-generated logic while the CPU is idling.
INT[13:0]	I	Maskable interrupts, 0 – 13. INT[13:0] are external interrupts that are prioritized and maskable. All INT inputs are level sensitive. All inputs are not synchronized to the core clock and feed directly to the interrupt-flag register (IFR). Synchronization and/or edge-detection circuits can be implemented outside of the core.
NMI	I	Nonmaskable interrupt. NMI is an external interrupt that cannot be masked. NMIn input is level-sensitive. The input is not synchronized to the core clock. Synchronization and/or edge-detection circuits can be implemented outside of the core.

† I =Input, O = Output.

Table A–4. Reset and Interrupt Signal Descriptions (Continued)

Signal Name	I/O†	Description
$\overline{RS}$	I	Synchronous reset. When $\overline{RS}$ is brought low, it causes the device to terminate execution, forcing internal-operating modes to a known state, and forcing the program to the RESET vector location (dependent on VMAP-signal input). When the signal is brought to a high level, the address located at the reset vector is fetched and written into the program counter (PC). Execution then begins at the vector location. An appropriate number of machine cycles are necessary to completely initialize internal states.
$\overline{SYSRS}$	O	System reset. $\overline{SYSRS}$ goes active (low) when $\overline{RS}$ is latched by the core. $\overline{SYSRS}$ goes high when the core has completed all internal-reset sequences and is ready to fetch the reset vector. $\overline{SYSRS}$ is synchronized to the rising edge of SYSCLKOUT.
VECT	O	Vector fetch. VECT is driven high for the duration of the reset and interrupt vector-fetch operation. VECT is valid on the rising edge of SYSCLKOUT and remains valid until the end of the program-read cycle. VECT can be used, in conjunction with the program-address-bus value, to fetch vectors from special interrupt-processing blocks external to the core.
VMAP	I	Vector map. VMAP is an input signal that selects the mapping of the reset and interrupt vectors. VMAP = 1 forces the vector table to high memory space starting at address 0x3FFFC0. VMAP = 0 forces the vector table to low memory space starting at address 0x000000. The VMAP signal is sampled on a reset, $\overline{RS}$, and the value mirrored in the VMAP bit in status register 1. The SETC/CLRC instructions can be used to modify the bit, thus remapping the vector table. This bit is accessible by the emulation hardware via the scan port and enables the test software to map the reset and interrupt vector table to low or high memory.
VMAPS	O	Vector-map bit status. VMAPS is an output signal that mirrors the status of the VMAP bit in status register 1.

† I =Input, O = Output.

A.1.5 Emulation Signals

Table A–5 describes the emulation signals.

Table A–5. Emulation Signal Descriptions

Signal Name	I/O†	Description
COREATPG	O	COREATPG is an output signal that goes active (high) when the ’C2700B0 core is put in automatic test pattern generation (ATPG) mode.
COREFTEST	O	COREFTEST is an output signal that goes active (high) when the ’C2700B0 core is put in functional test mode.
ET0I	I	Emulator input 0. ET0I is used by the emulator to force the CPU to trap to an emulator interrupt or to stop altogether. ET0I is also used for device operational mode control when TRSTn is low. ET0I is the output of an input buffer. The input side is the dedicated pin ET0.
ET0O	O	Emulator output 0. ET0O flags the emulator of an internally generated emulator event. ET0O can be configured to output various CPU events. ET0O is the input of an output buffer to form the dedicated pin ET0.
ET0Z	O	Emulator output 0 enable. ET0Z, when active (low), indicates ET0O is active. ETOZ is the 3-state enable of an output buffer to form the dedicated pin ET0.
ET1I	I	Emulator input 1. ET1I is used by the emulator to force the CPU to trap to an emulator interrupt or to stop altogether. This signal is also used for device operational mode control when TRSTn is low. ET1I is the output of an input buffer. The input side is the dedicated pin ET1.
ET1O	O	Emulator output 1. ET1O flags the emulator of an internally generated emulator event. ET1O can be configured to output varous CPU events. ET1O is the input of an output buffer.
ET1Z	O	Emulator output 1 enable. ET1Z, when active (low), indicates that ET1O is active. ET1Z is the 3-state enable of an output buffer.
MEMXFTEST	O	MEMXFTEST is an output signal that goes active (high) when the core is put in memory interface functional test mode.
PERISCANEN	O	Peripheral scan enable. PERISCANEN is used for peripheral ATPG and is connected to all the components.
PERIATPG	O	PERIATPG is an output signal that goes active (high) when the core is put in peripheral ATPG mode.
PERISCOUT	I	PERISCOUT is the scan out of the peripheral scan chain.
PERISCPATH	I	PERISCPATH is tied high or low depending on whether the peripheral scan chain is active.

† I = Input, O = Output
‡ IEEE Standard 1149.1–1990, IEEE Standard Test Access Port and Boundary-Scan Architecture

Table A–5. Emulation Signal Descriptions (Continued)

Signal Name	I/O†	Description
SLAVEIN	I	SLAVEIN is an input signal that puts the ’C2700B0 core in slave code and it must be brought out directly on the pins or it must be tied low.
TCK	I	Test clock. TCK is the input clock for the IEEE Standard 1149.1 (JTAG‡) serial scan operations and it is a free-running clock signal with a 50 percent duty cycle. The changes on test access port (TAP) input signals (TMS and TDI) are clocked into the TAP controller, instruction register, or selected test data register on the rising edge of TCK. Changes at the TAP output signal (TDO) occur on the falling edge of TCK. This signal must be brought to a dedicated pin of the device for test and emulation purposes.
TDI	I	Test data input. TDI is clocked into the selected register (instruction or data) on a rising edge of TCK. TDI must be brought out to a dedicated pin of the device for test and emulation purposes.
TDO	O	Test data out. The contents of the selected register (instruction or data) is shifted out of TDO on the falling edge of TCK. TDO is in high impedance state except when scanning of data is in progress. TDO must be brought to a dedicated pin of the device for test and emulation purposes.
$\overline{\text{TDOZ}}$	O	Emulator scan out enable. $\overline{\text{TDOZ}}$, when active (low), indicates TDO is active. It is connected to the 3-state control of the input/output buffer cell to form the dedicated pin TDO.
TMS	I	Test mode select. TMS control input is clocked into the TAP controller on the rising edge of TCK. If TMS is held high for five TCK cycles, the port shuts down and the device operates in its functional mode (i.e., the same effect as $\overline{\text{TRST}}$ low). TMS must be brought to a dedicated pin of the device for test and emulation purposes.
$\overline{\text{TRST}}$	I	Test reset. $\overline{\text{TRST}}$, when high, gives the scan system control of operations of the device. If this signal is driven low, the device operates in its functional mode and the test signals are ignored. This signal is asynchronous. If the device is taken to functional mode by clocking in a high TMS, $\overline{\text{TRST}}$ must be tied high. When $\overline{\text{TRST}}$ is low, ET0 and ET1 are inputs and indicate test modes of the device.
XLOGOFF	O	XLOGOFF is an output signal that must be connected to all of the memory interface conponents. XLOGOFF, when active (high), indicates that the CPU is either in slave mode or in core ATPG. The CPU outputs on the memory interface can contain invalid values.

† I = Input, O = Output
‡ IEEE Standard 1149.1–1990, IEEE Standard Test Access Port and Boundary-Scan Architecture

A.1.6 Visibility Port Signals

Table A–6 lists the visibility port signals.

CAUTION

Signals in Table A–6 are reserved for TI emulation use and must not be connected or used in a cDSP design.

Table A–6. Visibility Port Signals Descriptions

Signal Name	I/O†	Description
AEVTQUAL	I	Reserved
ANASTOP	I	Reserved
BREAKHI	I	Reserved
BREAKLO	I	Reserved
CSTOPPING	O	Reserved
CTOOLSACK	I	Reserved
DBGACCESSP	O	Reserved
DBGACCESSR	O	Reserved
DBGACCESSW	O	Reserved
DBGACK	O	Reserved
DBGM	O	Reserved
DBGPTYPE	O	Reserved
DBGRTYPE[1:0]	O	Reserved
DBGWTYPE	O	Reserved
DCON	O	Reserved
DEVTQUAL	I	Reserved
DFC	O	Reserved
EALLOW	O	Reserved
EXTTRGR	I	Reserved
EXTCNT0	I	Reserved

† I Input. Signal must be driven high or low to avoid uncontrolled power dissipation.
O Output. Output is always driven high or low.

Table A–6. Visibility Port Signals Descriptions (Continued)

Signal Name	I/O†	Description
EXTCNT1	I	Reserved
HERMIT	O	Reserved
MONPRIV	I	Reserved
MONPRIVO	O	Reserved
PSTAT	O	Reserved
PWRABORT	O	Reserved
RESERVED1	O	Reserved
RESERVED2	O	Reserved
RESETOUT	O	Reserved
RSTAT	O	Reserved
$\overline{\text{RTOSINT}}$	I	Reserved
TRACEHI	I	Reserved
TRACELO	I	Reserved
UBUS[31:0]	I	Reserved
USER0[3:0]	I	Reserved
USER1[3:0]	I	Reserved
VBANZ	O	Reserved
VCOND	O	Reserved
VDISCINSTR	O	Reserved
VDRDB[31:0]	O	Reserved
VHPI	O	Reserved
VINDRCT	O	Reserved
VINSTRJAM	O	Reserved
VIREG(31:0)	O	Reserved
VMAC	O	Reserved
VNEWINSTR	O	Reserved
VPAGE0	O	Reserved

† I Input. Signal must be driven high or low to avoid uncontrolled power dissipation.
O Output. Output is always driven high or low.

Table A–6. Visibility Port Signals Descriptions (Continued)

Signal Name	I/O†	Description
VPC	O	Reserved
VPCDISC	O	Reserved
VPIPEPROT	O	Reserved
VPRDB	O	Reserved
VPREAD	O	Reserved
VPWRITE	O	Reserved
VRESET	O	Reserved
VRPTINSTR	O	Reserved
VSPR	O	Reserved
VSPW	O	Reserved
WSTAT	O	Reserved

† I Input. Signal must be driven high or low to avoid uncontrolled power dissipation.
O Output. Output is always driven high or low.

A.2 CROM Wrapper Signals

Table A–7 describes the signals from the ’C2700B0 core to the CROM wrapper and from the CROM wrapper to the CROM core.

Table A–7. CROM Wrapper Signal Descriptions

Signal Name	I/O†	Description
ABORTREADY	I	CPU aborting signal from the ’C2700B0 core. When active (high), the CROM wrapper places its PRDB bus in the high-impedance state and deasserts the MEZ and MGZ signals. See ABORTREADY in Table A–2 on page A-11.
CLK	I	Clock signal from the ’C2700B0 core. This signal should also be connected from the CROM wrapper to the CLK input of the CROM core.
CPUSTAT	I	CPU status from the ’C2700B0 core. See CPUSTATP0/P1 in Table A–2 on page A-12.
DON	I	Data space enable. This signal is always tied high.
DRAB[21:0]	I	Data-read address bus from the ’C2700B0 core.
DRDB[31:0]	O	Data-read data bus to the ’C2700B0 core.
DRDS0	I	Data-read data-select low from the ’C2700B0 core.
DRDS1	I	Data-read data-select high from the ’C2700B0 core.
DRDYIN	I	Data-read data-ready in. Signal is daisy-chained from other wrapper/peripheral module DRDYOUT signal for multiple data memory blocks. If the CROM wrapper is first in the daisy chain, this signal is connected to ground.
DRDYOUT	O	Data-read data-ready out to the ’C2700B0 core. The CROM wrapper generates an active-high signal when it drives data on the DRDB bus.
DROREADY	O	Data-space memory-read output ready to the ’C2700B0 core. See DROREADY in Table A–2 on page A-12.
DSTRT[12:0]	I	Data-space start address bus used by the CRM wrapper for data-read address decoding. DSTRT(12:0) is compared with DRAB[21:9] to determine whether the current read is to the CROM wrapper. Low-order bits of DSTRT must be connected to the low-order bits of DRAB if the core is larger than 256 × 32 words. See section 2.2.1, *CROM Configuration Guidelines*, on page 2-4.
IFSTAT	I	Instruction-fetch status from the ’C2700B0 core.
MA[15:1]	O	Memory address output bus to the CROM core. This signal should be connected to the A inputs of the CROM core. Bit 0 is not used, because the CROM core is 32 bits wide. High-order bits must be unconnected if the core is smaller than 32K × 32 words.

† I = Input, O = Output.

Table A–7. CROM Wrapper Signal Descriptions (Continued)

Signal Name	I/O†	Description
MD[31:0]	I	Memory data in bus from the CROM core. This signal should be connected to the Q outputs of the CROM core.
MEMXFTEST	I	Memory functional test enable from the ’C2700B0 core. When asserted (high), the CROM wrapper overrides the MPNMC signal. See MEMXFTEST in Table A–5 on page A-17.
MEZ	O	Memory enable to the CROM core. Enables the CROM core for read operations. This signal should be connected to the EZ input of the CROM core.
MGZ	O	Memory output enable to the CROM core. When low, data appears at the memory output; when high, the outputs are in the high-impedance state. This signal should be connected to the GZ input of the CROM core.
MPNMC	I	Microprocessor/not microcontroller from your ASIC logic. This signal is ignored when the MEMXFTEST signal is high. When high, disables the CROM core from program memory space; when low, enables the CROM core in program memory space.
PAB[21:0]	I	Program address bus from the ’C2700B0 core.
PON	I	Program space enable. If this signal is tied high, the CROM wrapper is mapped to program space. If this signal is tied low, the CROM wrapper is mapped to data space.
POREADY	O	Program-space memory output ready to the ’C2700B0 core. See POREADY in Table A–2 on page A-13.
PRDB[31:0]	O	Program-read data bus to the ’C2700B0 core.
PRDS0	I	Program-read data-select low from the ’C2700B0 core.
PRDS1	I	Program-read data-select high from the ’C2700B0 core.
PRDYIN	I	Program-read data-ready in. Signal is daisy-chained from other wrapper/peripheral module PRDYOUT signal for multiple program memory blocks. If the CROM wrapper is first in the daisy chain, this signal is connected to ground.
PRDYOUT	O	Program-read data-ready out to the ’C2700B0 core. The CROM wrapper generates an active-high signal when it drives data on the PRDB bus. See PRDY in Table A–1 on page A-4.
PSTRT[12:0]	I	Program-space start address bus used by the CROM wrapper for program-space address decoding. PSTRT[12:0] is compared with PAB[21:9] to determine whether the current read is to the CROM wrapper. Low-order bits of PSTRT must be connected to the low-order bits of PAB if the core is larger than 256 × 32 words. See section 2.2.1, *CROM Configuration Guidelines*, on page 2-4.

† I = Input, O = Output.

Table A–7. CROM Wrapper Signal Descriptions (Continued)

Signal Name	I/O†	Description
ROMCLK	O	ROM core clock. When the WSTATE signal is low, this clock period is equal to CPU clock period; otherwise, this clock is an inverted divided-by-2 CPU clock and always starts new accesses with a low pulse.
SCIN	I	Scan input from the ’C2700B0 core. The scan pattern is applied to the CROM wrapper through this pin. See TDI in Table A–5 on page A-18.
SCEN	I	Scan enable from the ’C2700B0 core. Is used to place the PRDB bus in the high-impedance state when the CROM wrapper is being scanned. See PERISCANEN in Table A–5 on page A-17.
SCOUT	O	Scan output to the ’C2700B0 core. The pattern that is scanned from the CROM wrapper is obtained from this pin. See TDO in Table A–5 on page A-18.
SYSRSN	I	System reset from the ’C2700B0 core. When asserted (low), the CROM wrapper is reset. See SYSRS0 in Table A–4 on page A-15.
VECT	I	Interrupt vector fetch acknowledge. This signal is used in conjunction with the PIEENABLE signal. When both VECT and PIEENABLE signals are active, the program fetches from the CROM wrapper are disabled.
WSTATE	I	Wait state static tie off. When this signal is tied to logic low, the CROM wrapper assumes 0 wait-state accesses; when this signal is tied to logic high, the CROM wrapper assumes 1 wait-state accesses.
XLOGOFF	I	An automatic test pattern generation (ATPG) signal from the ’C2700B0 core. When active (high), the CROM wrapper places its PRDB bus in the high-impedance state and deasserts the MEZ and MGZ signals. See XLOGOFF in Table A–5 on page A-17.

† I = Input, O = Output.

A.3 SARAM Wrapper Signals

Table A–8 describes the signals from the ’C2700B0 core to the SARAM wrapper and from the SARAM wrapper to the SARAM core.

Table A–8. SARAM Wrapper Signal Descriptions

Signal Name	I/O†	Description
ABORTREADY	I	CPU aborting signal from the ’C2700B0 core. When active (high), the SARAM wrapper places its PRDB and DRDB buses in the high-impedance state. See ABORTREADY in Table A–2 on page A-11.
CLK	I	Clock signal from the ’C2700B0 core. This signal should also be connected from the SARAM wrapper to the CLK input of the SARAM cores.
CPUSTAT	I	CPU status from the ’C2700B0 core. See CPUSTATP0/P1 in Table A–2 on page A-12.
DON	I	Data space enable. This signal is always tied high.
DRAB[21:0]	I	Data-read address bus from the ’C2700B0 core. See DRAB in Table A–1 on page A-4.
DRDB[31:0]	O	Data-read data bus to the ’C2700B0 core. See DRDB in Table A–1 on page A-4.
DRDS0	I	Data-read data-select low from the ’C2700B0 core. See DRDS0 in A-4 on page .
DRDS1	I	Data-read data-select high from the ’C2700B0 core. See DRDS1 in Table A–1 on page A-4.
DRDYIN	I	Data-read data-ready in. Signal is daisy-chained from other wrapper/peripheral module DRDYOUT signal for multiple data memory blocks. If the SARAM wrapper is first in the daisy chain, this signal is connected to ground.
DRDYOUT	O	Data-read data-ready out to the ’C2700B0 core. The SARAM wrapper generates an active-high signal when it drives data on the DRDB bus. See DRDY in Table A–1 on page A-4.
DROREADY	O	Data-space memory-read output ready to the ’C2700B0 core. See DROREADY in Table A–2 on page A-12.
DRSTRT[11:0]	I	Data-space start address used by the SARAM wrapper for data-read address decoding. DRSTRT and DWSTRT must be mapped to the same address location. DRSTRT[11:0] is compared with DRAB[21:10] to determine whether the current read is to the SARAM wrapper. Low-order bits of DRSTRT must be connected to the low-order bits of DRAB if the core is larger than 512 × 32 words. See section 2.3.2, *SARAM Configuration Guidelines*, on page 2-13.

† I = Input, O = Output

Table A–8. SARAM Wrapper Signal Descriptions (Continued)

Signal Name	I/O†	Description
DWAB[21:0]	I	Data-write address bus from the ’C2700B0 core. See DWAB in Table A–1 on page A-7.
DWD[(31:0]	I	Data/program-write data bus from the ’C2700B0 core. See DWDB in Table A–1 on page A-4.
DWDS0	I	Data-write data-select low from the ’C2700B0 core. See DWDS0 in Table A–1 on page A-4.
DWDS1	I	Data-write data-select high from the ’C2700B0 core. See DWDS1 in Table A–1 on page A-4.
DWOREADY	O	Data-space memory-write output ready to the ’C2700B0 core. See DWOREADY in Table A–2 on page A-13.
DWSTRT[11:0]	I	Data-space start address bus used by the SARAM wrapper for data-write address decoding. DRSTRT and DWSTRT must be mapped to the same address location. DWSTRT[11:0] is compared with DWAB[21:10] to determine whether the current write is to the SARAM wrapper. Low-order bits of DWSTRT must be connected to the low-order bits of DWAB if the core is larger than 512 × 32 words. See section 2.3.2, *SARAM Configuration Guidelines*, on page 2-13.
MADD[14:1]	O	Memory address output bus to the SARAM cores. This signal should be connected to the A inputs of the SARAM cores. High-order bits should remain unconnected if the core is smaller than 32K × 16 words.
MCOREHIGH	O	Memory enable high to the upper bank of the 16-bit SARAM core. Enables the SARAM core for read operations. This signal should be connected to the EZ input of the SARAM core.
MCORELOW	O	Memory enable low to the lower bank of the 16-bit SARAM core. Enables the SARAM core for read operations. This signal should be connected to the EZ input of the SARAM core.
MDI[31:16]	I	Memory data in bus from the upper bank of the 16-bit SARAM core. This signal should be connected to the Q outputs of the SARAM core.
MDI[15:0]	I	Memory data in bus from the lower bank of the 16-bit SARAM core. This signal should be connected to the Q outputs of the SARAM core.
MDO[31:16]	O	Memory data out bus to the upper bank of the 16-bit SARAM core. This signal should be connected to the D inputs of the SARAM core.
MDO[15:0]	O	Memory data out bus to the lower bank of the 16-bit SARAM core. This should be connected to the D inputs of the SARAM core.

† I = Input, O = Output

Table A–8. SARAM Wrapper Signal Descriptions (Continued)

Signal Name	I/O†	Description
MEMXFTEST	I	Memory functional test enable from the ’C2700B0 core. When asserted (high), the SARAM wrapper overrides the MPNMC signal. See MEMXFTEST in Table A–5 on page A-17.
MRNW	O	Memory read-not-write enable to the SARAM cores. This signal should be connected to the WZ input of the SARAM cores. When high, enables the memory-read operation; when low, enables the memory-write operation.
MPNMC	I	Microprocessor/not microcontroller from your ASIC logic. This signal is ignored when the MEMXFTEST signal is high. When high, disables the SARAM core from program memory space; when low, enables the SARAM core in program memory space.
PAB[21:0]	I	Program address bus from the ’C2700B0 core. See PAB[21:0] in Table A–1 on page A-4.
PON	I	Program space enable. If this signal is tied high, the SARAM wrapper is mapped to program space. If this signal is tied low, the SARAM wrapper is mapped to data space.
POREADY	O	Program-space memory output ready to the ’C2700B0 core. See POREADY in Table A–2 on page A-13.
PRDB[31:0]	O	Program-read data bus to the ’C2700B0 core. See PRDB in Table A–1 on page A-4.
PRDS0	I	Program-read data-select low from the ’C2700B0 core. See PRDS0 in Table A–1 on page A-4.
PRDS1	I	Program-read data-select high from the ’C2700B0 core. See PRDS1 in Table A–1 on page A-4.
PRDYIN	I	Program-read data-ready in. Signal is daisy-chained from other wrapper/peripheral module PRDYOUT signal for multiple program memory blocks. If the SARAM wrapper is first in the daisy chain, this signal is connected to ground.
PRDYOUT	O	Program-read data-ready out to the ’C2700B0 core. The SARAM wrapper generates an active-high signal when it drives data on the PRDB bus. See PRDY in Table A–1 on page A-4.
PSTRT[11:0]	I	Program-space start address used by the SARAM wrapper for program-space address decoding. PSTRT[11:0] is compared with PAB[21:10] to determine whether the current read is to the SARAM wrapper. Low-order bits of PSTRT must be connected to the low-order bits of PAB if the core is larger than 512 × 32 words. See section 2.3.2, *SARAM Configuration Guidelines*, on page 2-13.
PWDS0	I	Program-write data-select low from the ’C2700B0 core. See PWDS0 in Table A–1 on page A-4.

† I = Input, O = Output

Table A–8. SARAM Wrapper Signal Descriptions (Continued)

Signal Name	I/O†	Description
PWDS1	I	Program-write data-select high from the ’C2700B0 core. See PWDS1 in Table A–1 on page A-4.
SCIN	I	Scan input from the ’C2700B0 core. The scan pattern is applied to the SARAM wrapper through this pin. See TDI in Table A–5 on page A-17.
SCEN	I	Scan enable from the ’C2700B0 core. Is used to place the PRDB and DRDB buses in the high-impedance state when the SARAM wrapper is being scanned. See PERISCANEN in Table A–5 on page A-17.
SCOUT	O	Scan output to the ’C2700B0 core. The pattern that is scanned from the SARAM wrapper is obtained from this pin. See TDO in Table A–5 on page A-17.
SYSRSN	I	System reset from the ’C2700B0 core. When asserted (low), the SARAM wrapper is reset. See SYSRS0 in Table A–4 on page A-15.
XLOGOFF	I	An automatic test pattern generation (ATPG) signal from the ’C2700B0 core. When active (high), the SARAM wrapper places its PRDB and DRDB buses in the high-impedance state. See XLOGOFF in Table A–5 on page A-17.

† I = Input, O = Output

A.4 External Interface (XINTF) Signals

The XINTF (Figure A–4) is a strobe-based interface similar to the standard logic interface bus on the T320C2xLP and TMS320C5x DSP cores. Table A–9 describes the signals of the XINTF.

Figure A–4. External Interface (XINTF) Signals Diagram

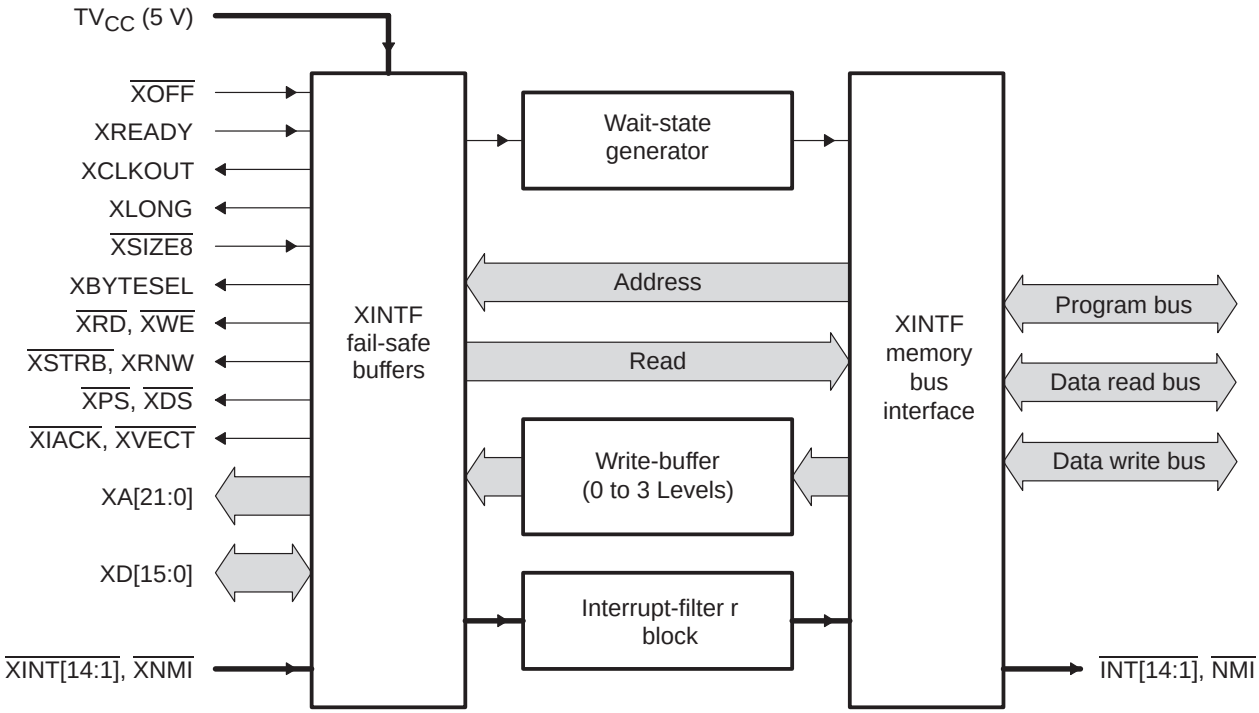


Table A–9. External Interface (XINTF) Signal Descriptions

Signal Name	I/O†	5 V/3 V‡	Description
XA[21:0]	O/Z	3.3 V	22-bit address bus
XBYTESEL	O/Z	3.3 V	Byte memory least significant address bit
XCLKOUT	O/Z	3.3 V	Divide by 1 or 2 of core clock
XCLKMODE	O/Z	3.3 V	Selects XCLKOUT divide-by-1 or divide-by-2 mode
XD[15:0]	I/O/Z	3.3 V	16-bit data bus
$\overline{\text{XDS}}$	O/Z	3.3 V	Data-space-select strobe
$\overline{\text{XEMUACC}}$	O/Z	3.3 V	Emulaton access strobe
XHOLD	I	3.3 V	External direct memory access (DMA) hold request
XHOLDA	I	3.3 V	External DMA hold acknowledge
XIACK	O/Z	3.3 V	Interrupt acknowledge
XIAQ	O/Z	3.3 V	Instruction acquisition strobe
$\overline{\text{XINT[14:1]}}$§	I	3.3 V	Maskable interrupts
XLONG	O/Z	3.3 V	Indicates 32-bit data access
XMPNMC§	I	3.3 V	Microprocessor/microcomputer mode select
XNMI§	I	3.3 V	Nonmaskable interrupt
XOFF§	I	3.3 V	Place all outputs in high-impedance mode
XPCDISC	O/Z	3.3 V	PC discontinuity strobe
XPS	O/Z	3.3 V	Program-space-select strobe
XREADY§	I	3.3 V	Input ready signal
XRD	O/Z	3.3 V	Read enable
XRNW	O/Z	3.3 V	Read/write select
XSIZE8§	I	3.3 V	Indicates peripheral supports 8-bit access or else defaults to 16-bit mode

† I = Input, O = Output, Z = High impedance
‡ All 5-V signals have a fail-safe buffer (FSB).
§ Use pullup on the XDS512.
¶ Use pulldown on the XDS512.
# Use pullup and jumper to ground.
Use pulldown internally on the ’2700-E3 chip.
◊ Use pullup internally on the ’2700-E3 chip.

Table A–9. External Interface (XINTF) Signal Descriptions (Continued)

Signal Name	I/O†	5 V/3 V‡	Description
XSTRB	O/Z	3.3 V	Active strobe signal
XVECT	O/Z	3.3 V	Vector fetch strobe
$\overline{\text{XWE}}$	O/Z	3.3 V	Write enable

† I = Input, O = Output, Z = High impedance
‡ All 5-V signals have a fail-safe buffer (FSB).
§ Use pullup on the XDS512.
¶ Use pulldown on the XDS512.
# Use pullup and jumper to ground.
Use pulldown internally on the ’2700-E3 chip.
◊ Use pullup internally on the ’2700-E3 chip.

A.5 Timer Signals

Table A–10 describes the signals from the ’C2700B0 core to the timer.

Table A–10. Timer Signal Descriptions

Signal Name	I/O†	5 V/3 V‡	Description
$\overline{\text{TINTO}}$	O/Z	3.3V	Timer 0 interrupt output
$\overline{\text{TINT1}}$	O/Z	3.3V	Timer 1 interrupt output

† I = Input, O = Output, Z = High impedance
‡ All 5-V signals have a fail-safe buffer (FSB).
§ Use pullup on the XDS512.
¶ Use pulldown on the XDS512.
# Use pullup and jumper to ground.
Use pulldown internally on the ’2700-E3 chip.
◊ Use pullup internally on the ’2700-E3 chip.

A.6 IEEE 1149.1 (JTAG) Signals

The TI debug interface uses the five standard IEEE 1149.1 (JTAG) signals: nTRST, TCK, TDI, TDO, and TMS, and the two TI extensions: EMU0 (ET0) and EMU1 (ET1). The 14-pin JTAG header used to interface the target to the scan controller also requires test clock out (TCKO), target supply (V_{DD}), and ground (GND). TCKO is a test clock signal from the scan controller to the target system that the target system uses if the target system does not supply its own test clock (if the target system does supply its own test clock, TCKO is not used). In many target systems, TCKO is just connected to TCK and used as the test clock. The 14-pin JTAG header is shown in Figure A–5. Table A–11 describes the signals.

Figure A–5. 14-Pin IEEE 1149.1 (JTAG) Header

TMS	1	2	nTRST
TDI	3	4	GND
PD (V_{DD})	5	6	no pin (key)
TDO	7	8	GND
TCK_RET	9	10	GND
TCK	11	12	GND
EMU0 (ET0)	13	14	EMU1 (ET1)

Table A–11. IEEE 1149.1 (JTAG) Header Interface Signal Descriptions

(a) JTAG Header Signals Directly Connected to the T320C2700B0 Core

Signal Name	I/O†	5 V/3 V‡	Description
EMU0◊ (ET0)	I/O/Z	3.3V	Emulation/test trigger channel 0
EMU1◊ (ET1)	I/O/Z	3.3V	Emulation/test trigger channel 1
$\overline{\text{TRST}}$	I	3.3V	JTAG test-logic reset
TCK	I	3.3V	JTAG test clock
TDI	I	3.3V	JTAG test data input
TDO	O/Z	3.3V	JTAG test data output
TMS	I	3.3V	JTAG test mode select

(b) JTAG Header Signals Not Directly Connected to the T320C2700B0 Core

Signal Name	I/O†	5 V/3 V‡	Description
GND	I/O/Z	3.3V	Ground
PD (V_{dd})	I	3.3V	Target power supply. This signal is used as a power detect.
TCK_RET	O/Z	3.3V	Test clock returned. This test clock signal is supplied by the scan controller to the application. This test clock signal can be used as the system TCK source or it can be ignored with the TCK source being generated by the target system.

† I = Input, O = Output, Z = High impedance
‡ All 5-V signals have a fail-safe buffer (FSB).
§ Use pullup on the XDS512.
¶ Use pulldown on the XDS512.
# Use pullup and jumper to ground.
Use pulldown internally on the ’2700-E3 chip.
◊ Use pullup internally on the ’2700-E3 chip.

Appendix B

T320C2700B0 Instruction Set Summary

This apendix presents a summary of the instruction set alphabetically and by operation type (arithmetic instructions, logical instructions, branch instructions, etc.) and explains how 32-bit accesses are aligned to even addresses. For a complete discussion of the instruction set syntax, see the *TMS320C27xx DSP CPU and Instruction Set Reference Guide*.

Topic **Page**

B.1 Instruction Set Summary Overview

The ’C2700B0 uses the following syntax formats. When a destination is specified, it is always the leftmost operand.

Format	Example
Mnemonic destination, source1, source2	`AND AH,@7,#16 ; Read 2 source values. ; Write logical AND of ; those values to destination.`
Mnemonic destination, source 1	`MOV ACC,@7 ; Read from source. Write ; value to destination.`
Mnemonic source1, source2	`CMPL ACC,@7 ; Compare two source values.`
Mnemonic source	`PUSH ST0 ; Read from source. Stack is ; destination.`
Mnemonic destination	`POP ST1 ; Source is stack. Write to ; destination.`

This appendix provides summaries of the instruction set as follows:

Summary Type	See ...	Comment
Alphabetical by mnemonic	Section B.2 on page B-3	This summary contains brief descriptions of what the instructions do.
By operation type	Section B.3 on page B-9	This summary contains information about instruction sizes (words) and cycle times. It also contains status bit information.

B.2 Alphabetical Instruction Set Summary by Mnemonic

Table B–1 lists the instructions in alphabetical order by mnemonic, and gives a brief description of the instruction and comments about the instruction.

Table B–1. Alphabetical Instruction Set Summary

Mnemonic	Description	Comments
ABORTI	Abort interrupt	Used when an interrupt will not be returned from in the normal manner
ABS	Absolute value of accumulator	Finds the absolute value of the full 32-bit accumulator
ADD	Add value to specified location	Adds a 16-bit value to a data-memory location or a register. This value can be from data-memory or a register, or it can be a constant.
ADDB	Add short value to specified register	Adds a 7-bit constant to an auxiliary register or the stack pointer, or adds an 8-bit constant to AH, AL, or ACC
ADDCU	Add unsigned value plus carry bit to accumulator	Unsigned value is a 16-bit data-memory or register value
ADDL	Add long value to accumulator	Adds a 32-bit data-memory value or a 22-bit auxiliary register value to ACC
ADDU	Add unsigned value to accumulator	Unsigned value is a 16-bit data-memory or register value
ADRK	Add to current auxiliary register	Adds an 8-bit constant to an auxiliary register
AND	Bitwise AND	Performs an AND operation on two 16-bit values Can be used to clear values in the interrupt enable register (IER) and the interrupt flag register (IFR)
ANDB	Bitwise AND with short value	Performs an AND operation on an 8-bit value and an accumulator half (AH or AL)
ASP	Align stack pointer	If the stack pointer (SP) points to an odd address, ASP increments SP by 1. This aligns SP to an even address.
ASR	Arithmetic shift right	Shifts AH or AL right by the amount you specify. During the shift, the value is sign extended.
B	Branch	Uses a 16-bit offset to perform an offset branch
BANZ	Branch if auxiliary register not equal to 0	Uses a 16-bit offset to perform an offset branch

Table B–1. Alphabetical Instruction Set Summary (Continued)

Mnemonic	Description	Comments
CALL	Call	Stores the return address to the stack and then performs an absolute branch to a specified 22-bit address
CLRC	Clear status bits	Clears one or more of certain bits in status registers ST0 and ST1
CMP	Compare	Uses a subtraction to compare two values. The result is not stored but is reflected by flag bits.
CMPB	Compare with short value	Uses a subtraction to compare AH or AL with an 8-bit constant. The result is not stored but is reflected by flag bits.
CMPL	Compare with long value	Uses a subtraction to compare ACC with a 32-bit data-memory value or a 22-bit auxiliary register value. The result is not stored but is reflected by flag bits.
DEC	Decrement specified value by 1	Acts on a 16-bit data-memory location or register
EALLOW	Allow access to emulation registers	Enables access to the memory-mapped emulation registers
EDIS	Disallow access to emulation registers	Disables access to the memory-mapped emulation registers
ESTOP0	Emulator software breakpoint	Used to create a software breakpoint
ESTOP1	Embedded software breakpoint	Used to create an embedded software breakpoint
FFC	Fast function call	Stores the return address to auxiliary register XAR7 and then performs an absolute branch to a specified 22-bit address
IACK	Interrupt acknowledge	Drives a specified 16-bit value on the low 16 bits of the data-write data bus, DWDB(15:0). Can be used in an interrupt service routine to inform external hardware that a certain interrupt is being serviced by the CPU.
IDLE	Idle until interrupt	Places CPU in a dormant state until it is awakened by an enabled or nonmaskable hardware interrupt
INC	Increment specified value by 1	Acts on a 16-bit data-memory location or register

Table B–1. Alphabetical Instruction Set Summary (Continued)

Mnemonic	Description	Comments
INTR	Software interrupt	Can be used to initiate maskable interrupts INT1–INT14, DLOGINT, RTOSINT, and NMI. For those interrupts assigned bits in the interrupt enable register (IER) and the interrupt flag register (IER), the bits are cleared.
IRET	Return from interrupt and restore register pairs	Restores the program counter (PC) value and other register values that were saved to the stack by an interrupt operation
ITRAP0	Instruction trap 0	Causes an illegal-instruction trap, as with a TRAP #19 instruction
ITRAP1	Instruction trap 1	Causes an illegal-instruction trap, as with a TRAP #19 instruction
LB	Long branch	Performs an absolute branch to a specified 22-bit address
LOOPNZ	Loop while not zero	Repeats until a specified test results in 0
LOOPZ	Loop while zero	Repeats until a specified test results in a nonzero value
LSL	Logical shift left	Shifts AH, AL, or ACC left by the amount you specify
LSR	Logical shift right	Shifts AH or AL right by the amount you specify. During the shift, the value is not sign extended.
MAC	Multiply and accumulate with preload to T register	Adds the P register value to ACC, loads T register and then multiplies T register value by value from program memory
MOV	Move value	Copies a value from one location to another, or loads a location with a specified value
MOVA	Load T register and add previous product to accumulator	Loads the T register and adds the P register value to ACC
MOVB	Move short value	Loads a register with an 8-bit constant or moves a specified byte from one location to another
MOVH	Store high word	Stores the high word of the P register or ACC to data-memory or to a register
MOVL	Move long value	Enables loads and stores that use ACC and 32-bit data-memory locations

Table B–1. Alphabetical Instruction Set Summary (Continued)

Mnemonic	Description	Comments
MOVP	Load T register and load previous product to accumulator	Loads the T register and stores the P register value to ACC
MOVS	Load T register and subtract previous product from accumulator	Loads the T register and subtracts the P register value from ACC
MOVU	Load accumulator with unsigned word	Loads AL with an unsigned 16-bit value and clears AH
MOVW	Load entire data page pointer	Loads the entire data page pointer (DP) with a 16-bit constant. One syntax of the MOV instruction enables you to load only the 10 LSBs of the DP.
MPY	Multiply	Multiplies a 16-bit value by another 16-bit value
MPYA	Multiply and accumulate previous product	Adds the P register value to ACC and then multiplies two 16-bit values
MPYB	Multiply signed value by unsigned short value	Multiplies a signed 16-bit value by an unsigned 8-bit value
MPYS	Multiply and subtract previous product	Subtracts the P register value from ACC and then multiplies a 16-bit value by another 16-bit value
MPYU	Unsigned multiply	Multiplies an unsigned 16-bit value by another unsigned 16-bit value
MPYXU	Multiply signed value by unsigned value	Multiplies a signed 16-bit value by an unsigned 16-bit value
NASP	Unalign stack pointer	Undoes a previous alignment of SP performed by the ASP instruction
NEG	Negative of accumulator value	Finds the negative of the value in AH, AL, or ACC
NOP	No operation	Can be used to purposely create inactive cycles. Can also be used to increment an auxiliary register or the stack pointer without performing any other task.
NORM	Normalize accumulator	Can be used to remove extra sign bits from a value in ACC
NOT	Complement of accumulator value	Finds complement of AH, AL, or ACC

Table B–1. Alphabetical Instruction Set Summary (Continued)

Mnemonic	Description	Comments
OR	Bitwise OR	Performs an OR operation on two 16-bit values Can be used to set values in the interrupt enable register (IER) and the interrupt flag register (IFR). If the AND instruction sets an IFR and the interrupt is enabled, the interrupt is serviced.
ORB	Bitwise OR with short value	Performs an OR operation on an 8-bit value and an accumulator half (AH or AL)
POP	Restore from stack	Copies a 16-bit value or a 32-bit register pair from the stack to a data-memory location or a register
PREAD	Read from program memory	Loads a 16-bit data-memory location or register with a value from program memory
PUSH	Save value on stack	Copies a 16-bit value or a 32-bit register pair to the stack
PWRITE	Write to program memory	Loads a program-memory location with a value from a 16-bit data-memory location or register
RET	Return	Loads the program counter (PC) from the stack
RETE	Return with interrupts enabled	Loads the program counter (PC) from the stack and clears the interrupt global mask bit (INTM). By clearing INTM, it enables maskable interrupts.
ROL	Rotate accumulator left	Can be seen as the left rotation of a 33-bit value that is the concatenation of the carry bit (C) and ACC.
ROR	Rotate accumulator right	Can be seen as the right rotation of a 33-bit value that is the concatenation of the carry bit (C) and ACC.
RPT	Repeat next instruction	Causes the following instruction to repeat a specified number of times
SAT	Saturate accumulator	Fills ACC with a saturation value that reflects the net overflow represented in the overflow counter (OVC)
SB	Short branch	Uses an 8-bit offset to perform an offset branch
SBBU	Subtract unsigned value plus inverse borrow from accumulator	Unsigned value is a 16-bit data-memory or register value
SBRK	Subtract from specified auxiliary register	Subtracts an 8-bit constant from an auxiliary register
SETC	Set status bits	Sets one or more of specified bits in status registers ST0 and ST1

Table B–1. Alphabetical Instruction Set Summary (Continued)

Mnemonic	Description	Comments
SFR	Shift accumulator right	Shifts ACC right by the amount you specify. During the shift, the value is sign extended if the sign-extension mode bit (SXM) is 1 or is not sign extended if SXM is 0.
SPM	Set product shift mode	Sets the product shift mode bits (PM), which determine how certain instructions shift the P register value
SUB	Subtract value from specified location	Subtracts a 16-bit value from a data-memory location or a register. This value can be from data-memory or a register, or can be a constant.
SUBB	Subtract short value from specified register	Subtracts a 7-bit constant from an auxiliary register or the stack pointer, or subtracts an 8-bit constant from ACC
SUBCU	Conditional subtraction	Used for 16-bit division
SUBL	Subtract long value from accumulator	Subtracts a 32-bit data-memory value or a 22-bit auxiliary register value from ACC
SUBU	Subtract unsigned value from accumulator	Unsigned value is a 16-bit data-memory or register value
SXTB	Sign extend least significant byte of accumulator half	Sign extends the least significant byte of AH or AL
TBIT	Test specified bit	Tests a specified bit of a 16-bit data-memory location or register. The value of the bit is reflected by the test/control flag bit (TC).
TEST	Test for accumulator equal to zero	Uses a subtraction to compare ACC with 0. The result is not stored, but is reflected by flag bits.
TRAP	Software trap	Can be used to initiate any interrupt. Unlike the INTR instruction, the TRAP instruction never affects bits in the IER and IFR.
XOR	Bitwise exclusive OR	Performs an exclusive OR operation on two 16-bit values
XORB	Bitwise exclusive OR with short value	Performs an exclusive OR operation on an 8-bit value and an accumulator half (AH or AL)

B.3 Instruction Set Summary by Operation Type

Table B–2 through Table B–13 provide a summary of the instruction syntaxes according to the following functional groups:

- ❑ Address register operations (see Table B–2 on page B-10)
- ❑ Push and pop stack operations (see Table B–3 on page B-10)
- ❑ AX (AH, AL) operations (see Table B–4 on page B-12)
- ❑ AX (AH, AL) byte operations (see Table B–5 on page B-13)
- ❑ ACC operations (see Table B–6 on page B-13)
- ❑ ACC 32-bit operations (see Table B–7 on page B-15)
- ❑ Operations on memory or register (see Table B–8 on page B-15)
- ❑ Data move operations (see Table B–9 on page B-16)
- ❑ Program flow operations (see Table B–10 on page B-16)
- ❑ Math operations (see Table B–11 on page B-17)
- ❑ Control operations (see Table B–12 on page B-18)
- ❑ Emulation operations (see Table B–13 on page B-19)

The cycle times shown are the minimum cycle times.

Table B–2. Address Register Operations (AR0–AR7, XAR6, XAR7, DP, SP)

Instruction Syntax		Words	Cycles	Status Bits: Affected By	Status Bits: Affects
ADDB	*aux*, *#7bit*	1	1	–	–
ADDB	SP, *#7bit*	1	1	–	–
ADRK	*#8bit*	1	1	ARP	–
MOV	AR*x*, *loc*	1	1	–	–
MOV	XAR*n*, *locLong*	1	1	–	–
MOV	XAR*n*, *#22bit*	2	1	–	–
MOV	DP, *#10bit*	1	1	–	–
MOV	*loc*, AR*x*	1	1	–	N, Z
MOV	*locLong*, XAR*n*	1	1	–	–
MOVB	AR*x*, *#8bit*	1	1	–	–
MOVW	DP, *#16bit*	2	1	–	–
SBRK	*#8bit*	1	1	ARP	–
SUBB	*aux*, *#7bit*	1	1	–	–
SUBB	SP, *#7bit*	1	1	–	–

Table B–3. Push and Pop Stack Operations

Instruction Syntax		Words	Cycles	Status Bits: Affected By	Status Bits: Affects
POP	*loc*	1	2	–	N, Z
POP	DBGIER	1	5	–	–
POP	DP	1	1	–	–
POP	ST0	1	1	–	C, N, OVM, OVC, PM, SXM, TC, V, Z
POP	ST1	1	5	–	ARP, DBGM, EALLOW, INTM, PAGE0, SPA, VMAP
POP	T:ST0	1	1	–	C, N, OVM, OVC, PM, SXM, TC, V, Z

Table B–3. Push and Pop Stack Operations (Continued)

Instruction Syntax		Words	Cycles	Status Bits	
				Affected By	Affects
POP	DP:ST1	1	5	–	ARP, DBGM, EALLOW, INTM, PAGE0, SPA, VMAP
POP	PH:PL	1	1	–	–
POP	AR1:AR0	1	1	–	–
POP	AR3:AR2	1	1	–	–
POP	AR5:AR4	1	1	–	–
POP	XAR*n*	1	1	–	–
PUSH	*loc*	1	2	–	–
PUSH	DBGIER	1	1	–	–
PUSH	DP	1	1	–	–
PUSH	IFR	1	1	–	–
PUSH	ST0	1	1	–	–
PUSH	ST1	1	1	–	–
PUSH	T:ST0	1	1	–	–
PUSH	DP:ST1	1	1	–	–
PUSH	PH:PL	1	1	–	–
PUSH	AR1:AR0	1	1	–	–
PUSH	AR3:AR2	1	1	–	–
PUSH	AR5:AR4	1	1	–	–
PUSH	XAR*n*	1	1	–	–

Table B–4. AX (AH, AL) Operations

Instruction Syntax		Words	Cycles	Status Bits	
				Affected By	Affects
ADD	A*X*, *loc*	1	1	–	C, N, V, Z
ADDB	A*X*, *#8BitSigned*	1	1	–	C, N, V, Z
AND	A*X*, *loc*	1	1	–	N, Z
AND	A*X*, *loc*, *#16BitMask*	2	1	–	N, Z
ANDB	A*X*, *#8BitMask*	1	1	–	N, Z
ASR	A*X*, *shift*	1	1	–	C, N, Z
ASR	A*X*, T	1	1	–	C, N, Z
CMP	A*X*, *loc*	1	1	–	C, N, Z
CMPB	A*X*, *#8bit*	1	1	–	C, N, Z
LSL	A*X*, *shift*	1	1	–	C, N, Z
LSL	A*X*, T	1	1	–	C, N, Z
LSR	A*X*, *shift*	1	1	–	C, N, Z
LSR	A*X*, T	1	1	–	C, N, Z
MOV	A*X*, *loc*	1	1	–	N, Z
MOV	*loc*, A*X*	1	1	–	N, Z
MOVB	A*X*, *#8bit*	1	1	–	N, Z
NEG	A*X*	1	1	–	C, N, V, Z
NOT	A*X*	1	1	–	N, Z
OR	A*X*, *loc*	1	1	–	N, Z
ORB	A*X*, *#8BitMask*	1	1	–	N, Z
SUB	A*X*, *loc*	1	1	–	C, N, V, Z
XOR	A*X*, *loc*	1	1	–	N, Z
XORB	A*X*, *#8BitMask*	1	1	–	N, Z

Table B–5. AX (AH, AL) Byte Operations

Instruction Syntax		Words	Cycles	Status Bits: Affected By	Status Bits: Affects
MOVB	A*X*.LSB, *loc*	1	1	–	N, Z
MOVB	*loc*, A*X*.LSB	1	1	–	N, Z
MOVB	A*X*.MSB, *loc*	1	1	–	N, Z
MOVB	*loc*, A*X*.MSB	1	1	–	N, Z
SXTB	A*X*	1	1	–	N, Z

Table B–6. ACC Operations

Instruction Syntax		Words	Cycles	Status Bits: Affected By	Status Bits: Affects
ABS	ACC	1	1	OVM	C, N, V, Z
ADD	ACC, *loc* << *shift*	1	1	OVM, SXM	C, N, V, Z
ADD	ACC, #*16BitSU* << *shift*	2	1	OVM, SXM	C, N, V, Z
ADD	ACC, P	1	1	OVM, PM	C, N, V, Z
ADDB	ACC, #*8bit*	1	1	OVM	C, N, V, Z
ADDCU	ACC, *loc*	1	1	OVM	C, N, V, Z
ADDU	ACC, *loc*	1	1	OVM	C, N, V, Z
LSL	ACC, *shift*	1	1	–	C, N, Z
LSL	ACC, T	1	1	–	C, N, Z
MOV	ACC, *loc* << *shift*	1	1	SXM	N, Z
MOV	ACC, #*16BitSU* << *shift*	2	1	SXM	N, Z
MOV	ACC, P	1	1	PM	N, Z
MOV	*loc*, ACC << *shift*	1	1	–	N, Z
MOVB	ACC, #*8bit*	1	1	–	N, Z
MOVH	*loc*, ACC << *shift*	1	1	–	N, Z
MOVU	ACC, *loc*	1	1	–	N, Z
NEG	ACC	1	1	OVM	C, N, V, Z

Table B–6. ACC Operations (Continued)

Instruction Syntax		Words	Cycles	Status Bits: Affected By	Status Bits: Affects
NORM	ACC, *aux*++	1	4†	–	N, TC, Z
NORM	ACC, *aux*–––	1	4†	–	N, TC, Z
NOT	ACC	1	1	–	N, Z
ROL	ACC	1	1†	–	C, N, Z
ROR	ACC	1	1†	–	C, N, Z
SAT	ACC	1	1	OVC	C, N, OVC, V, Z
SBBU	ACC, *loc*	1	1	OVM	C, N, V, Z
SFR	ACC, *shift*	1	1	SXM	C, N, Z
SFR	ACC, T	1	1	SXM	C, N, Z
SUB	ACC, *loc* << *shift*	1	1	OVM, SXM	C, N, V, Z
SUB	ACC, #*16BitSU* << *shift*	2	1	OVM, SXM	C, N, V, Z
SUB	ACC, P	1	1	OVM, PM	C, N, V, Z
SUBB	ACC, #*8bit*	1	1	OVM	C, N, V, Z
SUBCU	ACC, *loc*	1	1†	–	C, N, OVC, V, Z
SUBU	ACC, *loc*	1	1	OVM	C, N, V, Z
TEST	ACC	1	1	–	N, Z

† This instruction is repeatable. The number of cycles given is for nonrepeated execution.

Table B–7. ACC 32-Bit Operations

Instruction Syntax		Words	Cycles	Status Bits: Affected By	Status Bits: Affects
ADDL	ACC, *locLong*	1	1	OVM	C, N, V, Z
CMPL	ACC, *locLong*	1	1	–	C, N, Z
MOVL	ACC, *locLong*	1	1	–	N, Z
MOVL	*locLong*, ACC	1	1	–	–
SUBL	ACC, *locLong*	1	1	OVM	C, N, V, Z

Table B–8. Operations on Memory or Register

Instruction Syntax		Words	Cycles	Status Bits: Affected By	Status Bits: Affects
ADD	*loc*, A*X*	1	1	–	C, N, V, Z
ADD	*loc*, *#16BitSigned*	2	1	–	C, N, V, Z
AND	*loc*, A*X*	1	1	–	N, Z
AND	*loc*, *#16BitMask*	2	1	–	N, Z
CMP	*loc*, *#16BitSigned*	2	1	–	C, N, Z
DEC	*loc*	1	1	–	C, N, V, Z
INC	*loc*	1	1	–	C, N, V, Z
LOOPNZ	*loc*, *#16BitMask*	2	5	–	LOOP, N, Z
LOOPZ	*loc*, *#16BitMask*	2	5	–	LOOP, N, Z
OR	*loc*, A*X*	1	1	–	N, Z
OR	*loc*, *#16BitMask*	2	1	–	N, Z
SUB	*loc*, A*X*	1	1	–	C, N, V, Z
TBIT	*loc*, *#BitNumber*	1	1	–	TC
XOR	*loc*, A*X*	1	1	–	N, Z
XOR	*loc*, *#16BitMask*	2	1	–	N, Z

Table B–9. Data Move Operations

Instruction Syntax		Words	Cycles	Status Bits: Affected By	Status Bits: Affects
MOV	*(0:*16bit*), *loc*	2	2†	–	–
MOP	*loc*, #0	1	1†	–	N, Z
MOV	*loc*, #*16bit*	2	1†	–	N, Z
MOV	*loc*, *(0:*16bit*)	2	2†	–	N, Z
PREAD	*loc*, *XAR7	1	2†	–	N, Z
PWRITE	*XAR7, *loc*	1	5†	–	–

† This instruction is repeatable. The number of cycles given is for nonrepeated execution.

Table B–10. Program Flow Operations

Instruction Syntax		Words	Cycles	Status Bits: Affected By	Status Bits: Affects
B	*16BitOffset*, *cond*	2	4/7†	–	V (if tested)
BANZ	*16BitOffset*, AR*x*––	2	2/4†	–	–
CALL	*22BitAddress*	2	4	–	–
CALL	*XAR7	1	4	–	–
FFC	XAR7, *22BitAddress*	2	4	–	–
IRET		1	8	–	ARP, C, DBGM, INTM, N, OVC, OVM, PAGE0, PM, SPA, SXM, TC, V, VMAP, Z
LB	*22BitAddress*	2	4	–	–
LB	*XAR7	1	4	–	–
RET		1	8	–	–
RETE		1	8	–	INTM
SB	*8BitOffset*, *cond*	1	4/7†	–	V (if tested)

† X/Y: X cycles are required if the branch is not taken. Y cycles are required if the branch is taken.

Table B–11. Math Operations

Instruction Syntax		Words	Cycles	Status Bits: Affected By	Status Bits: Affects
CMP	ACC, P	1	1	PM	C, N, Z
MAC	P, *loc*, 0:*pmem*	2	2†	OVM, PM	C, N, V, Z
MOV	PH, *loc*	1	1	–	–
MOV	PL, *loc*	1	1	–	–
MOV	P, ACC	1	1	–	–
MOV	T, *loc*	1	1	–	–
MOV	*loc*, P	1	1	PM	N, Z
MOV	*loc*, T	1	1	–	N, Z
MOVA	T, *loc*	1	1	OVM, PM	C, N, V, Z
MOVH	*loc*, P	1	1	PM	N, Z
MOVP	T, *loc*	1	1	PM	N, Z
MOVS	T, *loc*	1	1	OVM, PM	C, N, V, Z
MPY	ACC, *loc*, *#16BitSigned*	2	1	–	N, Z
MPY	ACC, T, *loc*	1	1	–	N, Z
MPY	P, T, *loc*	1	1	–	–
MPYA	P, *loc*, *#16BitSigned*	2	1	OVM, PM	C, N, V, Z
MPYA	P, T, *loc*	1	1	OVM, PM	C, N, V, Z
MPYB	ACC, T, *#8bit*	1	1	–	N, Z
MPYB	P, T, *#8bit*	1	1	–	–
MPYS	P, T, *loc*	1	1	OVM, PM	C, N, V, Z
MPYU	ACC, T, *loc*	1	1	–	N, Z
MPYU	P, T, *loc*	1	1	–	–
MPYXU	ACC, T, *loc*	1	1	–	N, Z
MPYXU	P, T, *loc*	1	1	–	–

† This instruction is repeatable. The number of cycles given is for nonrepeated execution.

Table B–12. Control Operations

Instruction Syntax		Words	Cycles	Status Bits: Affected By	Status Bits: Affects
AND	IER, *#16BitMask*	2	2	–	–
AND	IFR, *#16BitMask*	2	2	–	–
ASP		1	1	SPA	SPA
CLRC	*BitName1* { , *BitName2*}	1	1 or 2†	–	Specified status bit(s): C, DBGM, INTM, OVM, PAGE0, SXM, TC, VMAP
CLRC	*8BitMask*	1	1 or 2†	–	Specified status bit(s): C, DBGM, INTM, OVM, PAGE0, SXM, TC, VMAP
IACK	#*VectorValue*	2	1	–	–
IDLE		1	5	–	IDLESTAT
INTR	INT*i*	1	8	–	DBGM, EALLOW, IDLESTAT, INTM, LOOP
INTR	DLOGINT	1	8	–	DBGM, EALLOW, IDLESTAT, INTM, LOOP
INTR	RTOSINT	1	8	–	DBGM, EALLOW, IDLESTAT, INTM, LOOP
INTR	NMI	1	8	–	DBGM, EALLOW, IDLESTAT, INTM, LOOP
MOV	IER, *loc*	1	5	–	–
MOV	*loc*, IER	1	1	–	–
NASP		1	1	SPA	SPA
NOP		1	1‡	–	–
NOP	**ind*	1	1‡	–	–
OR	IER, *#16BitMask*	2	2	–	–
OR	IFR, *#16BitMask*	2	2	–	–
RPT	*loc*	1	4	–	–

† If CLRC or SETC is to affect the INTM bit and/or the DBGM bit, the instruction requires two cycles; otherwise the instruction requires one cycle.
‡ This instruction is repeatable. The number of cycles given is for nonrepeated execution.

Table B–12. Control Operations (Continued)

Instruction Syntax		Words	Cycles	Status Bits: Affected By	Status Bits: Affects
RPT	*#8bit*	1	1	–	–
SETC	*BitName1* { , *BitName2*}	1	1 or 2†	–	Specified status bit(s): C, DBGM, INTM, OVM, PAGE0, SXM, TC, VMAP
SETC	*8BitMask*	1	1 or 2†	–	Specified status bit(s): C, DBGM, INTM, OVM, PAGE0, SXM, TC, VMAP
SPM	*ShiftMode*	1	1	–	PM
TRAP	*#VectorNumber*	1	8	–	DBGM, EALLOW, IDLESTAT, INTM, LOOP

† If CLRC or SETC is to affect the INTM bit and/or the DBGM bit, the instruction requires two cycles; otherwise the instruction requires one cycle.
‡ This instruction is repeatable. The number of cycles given is for nonrepeated execution.

Table B–13. Emulation Operations

Instruction Syntax	Words	Cycles	Status Bits: Affected By	Status Bits: Affects
ABORTI	1	2	–	DBGM
EALLOW	1	4	–	EALLOW
EDIS	1	4	–	EALLOW
ESTOP0	1	1	–	–
ESTOP1	1	1	–	–
ITRAP0	1	8	–	DBGM, EALLOW, IDLESTAT, INTM, LOOP
ITRAP1	1	8	–	DBGM, EALLOW, IDLESTAT, INTM, LOOP

B.4 Alignment of 32-Bit Accesses to Even Addresses

The ’C2700B0 core expects memory wrappers and peripheral-interface logic to align any 32-bit data read or write to an even address. If the address-generation logic generates an odd address, the memory wrapper or peripheral interface must begin reading or writing at the previous even address. This alignment does not affect the address values generated by the address-generation logic.

Consider Example B–1, which shows a 32-bit read from data memory. This example uses DP direct addressing mode. The data page pointer and the offset point to an odd address, 00 0085h. However, the CPU must receive the first word from an even address. It reads from addresses 00 0084h and 00 0085h. Then it loads the word at 00 0084h to the low half of ACC and the word at 00 0085h to the high half of ACC.

Example B–1. 32-Bit Read From Data-Memory

```
MOVL   ACC, @5   ; DP = 2. Offset = 5. Address = 000085
                 ; Load ACC with 32–bit value at
                 ; addressed 32-bit location.
```

Before instruction

DP	0002
ACC	0000 0000

Data memory

000084	4444
000085	5555
000086	6666

After instruction

DP	0002
ACC	5555 4444

Data memory

000084	4444
000085	5555
000086	6666

Example B–2 shows a 32-bit write. AR0 points to the odd address 00 0085h. Due to alignment, the low half of ACC is saved at address 00 0084h and the high half of ACC is saved at address 00 0085h. AR0 is not modified; it still holds 85h after the operation.

Example B–2. 32-Bit Write to Data Memory

```
MOVL   *+AR0[0], ACC   ; AR0 = 85
                       ; Save 32–bit ACC to 32–bit location
                       ; pointed to by AR0.
```

Before instruction

AR0	0085
ACC	aaaa bbbb

Data memory

000084	4444
000085	5555
000086	6666

After instruction

AR0	0085
ACC	aaaa bbbb

Data memory

000084	bbbb
000085	aaaa
000086	6666

Table B–14 describes the mechanisms that generate 32-bit-wide data accesses.

Table B–14. Mechanisms That Generate 32-Bit-Wide Data Accesses

Mechanism	32-Bit Operation(s)
Interrupt initiated by hardware or software (OR, INTR, or TRAP instruction).	Multiple register pairs are automatically saved on the stack. Each pair is saved in a single 32-bit store operation.
IRET instruction	Multiple register pairs are automatically restored from the stack. Each pair is restored in a single 32-bit load operation.
Any of the following syntaxes of the PUSH instruction: PUSH T:ST0 PUSH AR1:AR0 PUSH DP:ST1 PUSH AR3:AR2 PUSH PH:PL PUSH AR5:AR4 PUSH XAR*n*	A register pair is saved on the stack in a single 32-bit store operation.
Any of the following syntaxes of the POP instruction: POP T:ST0 POP AR1:AR0 POP DP:ST1 POP AR3:AR2 POP PH:PL POP AR5:AR4 POP XAR*n*	A register pair is restored from the stack in a single 32-bit load operation.

Either of the following syntaxes of the MOV instruction:
MOV *locLong*, XAR*n*
MOV XAR*n*, *locLong*
	When XAR6 or XAR7 is the source, this 22-bit value is stored in a single 32-bit store operation. The 6 MSBs of the value are stored as 0s.
	When XAR6 or XAR7 is loaded, 32 bits are read from data memory. The 22 LSBs are loaded to the auxiliary register.

MOVL instruction	ACC is loaded or stored using a single 32-bit operation.
ADDL, CMPL, or SUBL instruction	A 32-bit value is read from data memory before being added or subtracted from ACC.

Appendix C

DSPnetGEN Tutorial

This tutorial introduces the DSPnetGEN tool and illustrates the use of the tool by providing a step-by-step guide to building an example subcircuit for the ’C2700B0 device. The goal of this tutorial is to familiarize you with the building process.

This appendix assumes you are familiar with the ’C2700B0 cDSP architecture and memory interface.

For more information on the ’C2700B0 architecture, see the *TMS320C27xx DSP CPU and Instruction Set Reference Guide*. For more information on the ’C2700B0 memory interface, see the *T320C2700 Customizable Digital Signal Processor (cDSP) Core* data sheet and the *T320C2700B0 Customizable Digital Signal Processor (cDSP) Core (TSC6000 ASIC Libraries)* data sheet.

Topic **Page**

C.1 Introduction to DSPnetGEN

DSPnetGEN is a graphical user interface (GUI) that captures high level design specifications and writes out a structural netlist for simulation and synthesis. It is a useful tool for creating a netlist for the ’C2700B0 core, memory, and XINTF parts of a design.

DSPnetGEN eliminates the tedious connection issues involved with a complex design. DSPnetGEN uses the golden netlist files (GNF) to determine how to connect each module together. The ’C2700B0 GNF files have already been developed and are included in the design kit.

C.2 Specifying the Design

Before you can begin building a subdesign you must identify your design specifications. To illustrate, design a subcircuit consisting of the following specifications:

- A module inventory consisting of these modules:
  - 1 T320C2700B0 DSP core processor
  - 3 SRAMW RAM wrappers
  - 1 MVSRAMW RAM wrapper
  - 1 CROMW ROM wrapper
  - 1 XINTF external interface
  - 1 CTSBUF clock-tree synthesis buffer
  - 2 MK00512032080 single port bit writable clocked ACE RAM (1K x 16)
  - 2 MR01024016084 single port clocked ACE RAM (each 1K X16)
  - 1 MV00512032081 single pot bit writable clocked ACE RAM (1K X 16)
  - 1 MX01024032080 ACE clocked ROM (2K X 16)
- A memory map consisting of modules listed in Table C–1:

Table C–1. Example Subdesign Memory Map

Module (Instance)	Program Space	Data Space	Size	Wait States
SRAMWIO (B0)	0x0	x400	1K X 16	0
SRAMWI1 (B1)	–	0x0	1K X16	0
SRAMWI2	0x1000	0x1000	2K X16	0
MVSRAMWI0	0x1800	0x1800	1K X 16	2
CROMWI0	03ff800	–	2K X 16	1
XINTFI0	0x2000	0x2000	8K X 16	–

- A ’2700B0 interrupt map as shown in Table C–2:

Table C–2. Example Subdesign Interrupt Map

T320C2700B0I0 Interrupts	Sources
.INTN[0]	XINTFI0.xifNBG2intn[0]
.INTN[10]	XINTFI0.xifNBG2intn[10]
.INTN[11]	XINTFI0.xifNBG2intn[11]
.INTN[12]	XINTFI0.xifNBG2intn[12]
.INTN[13]	XINTFI0.xifNBG2intn[13]
.INTN[1]	XINTFI0.xifNBG2intn[1]
.INTN[2]	XINTFI0.xifNBG2intn[2]
.INTN[3]	XINTFI0.xifNBG2intn[3]
.INTN[4]	XINTFI0.xifNBG2intn[4]
.INTN[5]	XINTFI0.xifNBG2intn[5]
.INTN[6]	XINTFI0.xifNBG2intn[6]
.INTN[7]	XINTFI0.xifNBG2intn[7]
.INTN[8]	XINTFI0.xifNBG2intn[8]
.INTN[9]	XINTFI0.xifNBG2intn[9]

- Configurable parameters of the protection bits listed as follows:
  - A PROT_RANGE value of 2K
  - A PROT_START_ADD value of 0x001000
  - PROT_ENABLE ON

- Configurable parameters of the XINTF listed in Table C–3:

Table C–3. Example Subdesign Configurable Parameters of the XINTF

XINTF Zone	Memory Space (Program and Data Space)	Start Address	Masksize
0	BOTH	0x002000	8K
1	NONE	0x004000	16K
2	NONE	0x008000	16K
3	NONE	0x00c000	16K
4	NONE	0x100000	1M
5	NONE	0x200000	1M
6	NONE	0x300000	512K
7	NONE	0x380000	512K

C.3 Starting DSPnetGEN

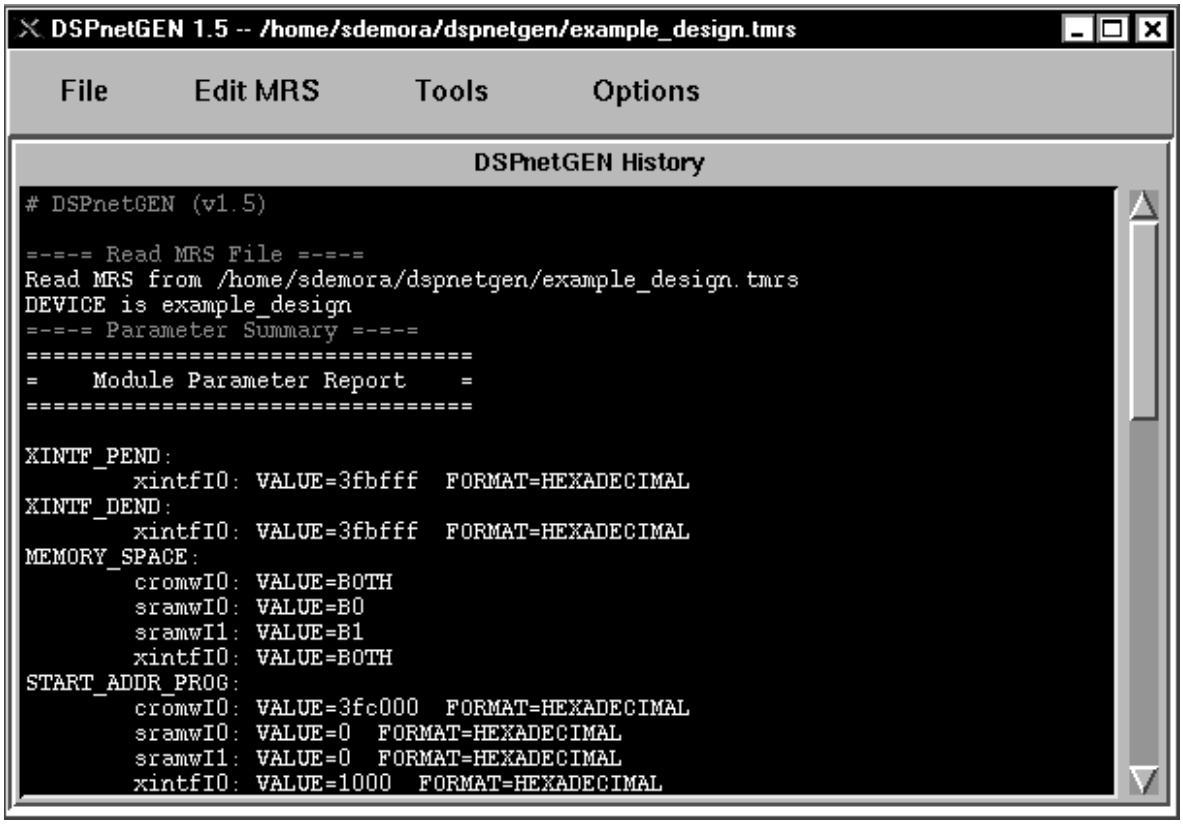
To start DSPnetGEN, you must first invoke DSPnetGEN from your UNIX prompt and then create a new Machine Readable Specification (MRS).

C.3.1 Invoking DSPnetGEN

To invoke DSPnetGEN, enter the following command on your UNIX command line:

```
dspnetgen &
```

The DSPnetGEN History window is displayed when you invoke DSPnetGEN.

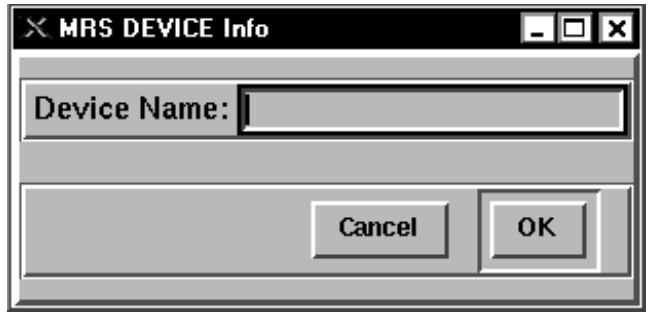


C.3.2 Creating a New Machine Readable Specification

The purpose of the DSPnetGEN user interface is to assist you with the creation of a machine readable specification (MRS) for the device being designed.

To capture a new design, follow these steps:

1) From the File menu on the DSPnetGEN History window, select New to display the MRS DEVICE Info dialog box.



2) In the Device Name field, enter a name for your device.

   You may enter any name for the subdesign in the Device Name field. For convenience, call it example_design.

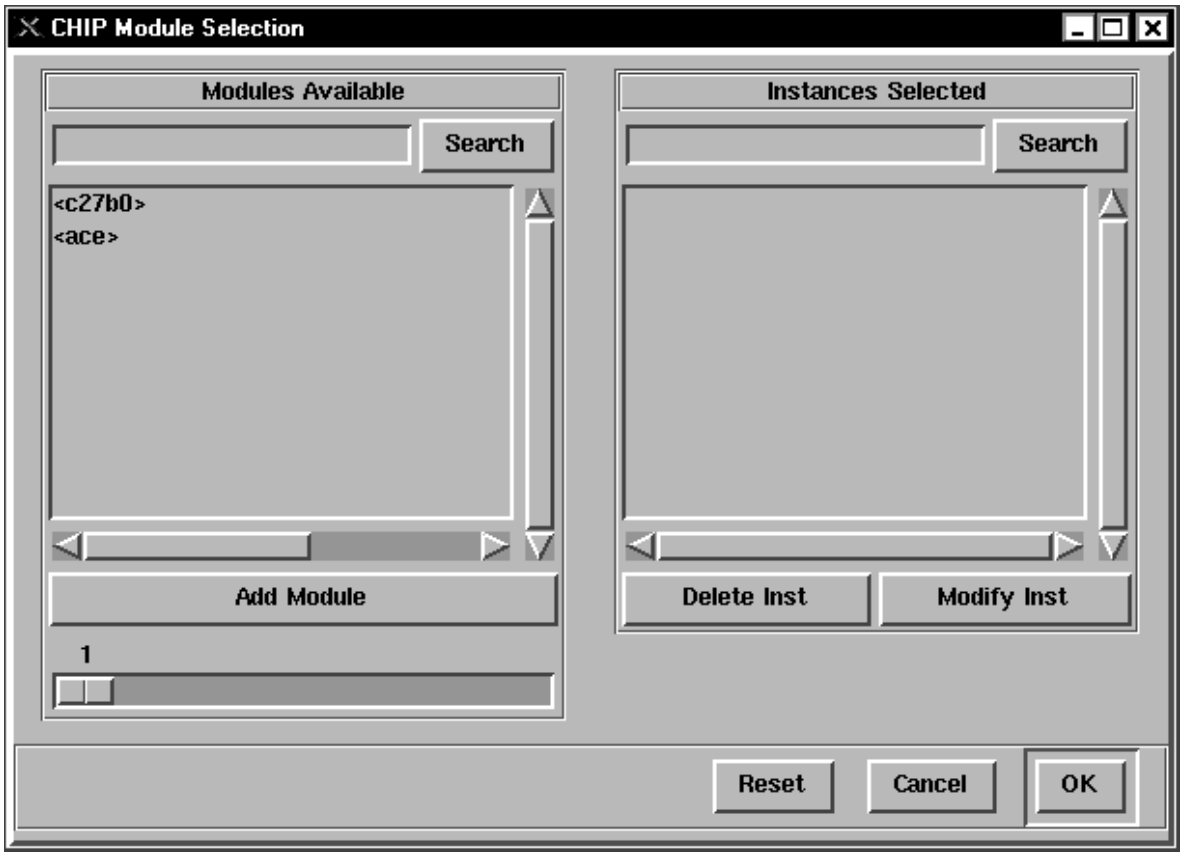
3) Click OK.

C.4 Selecting Modules

This step allows you to specify which library modules to include in your design.

To specify which library modules to place in the example_design, follow these steps:

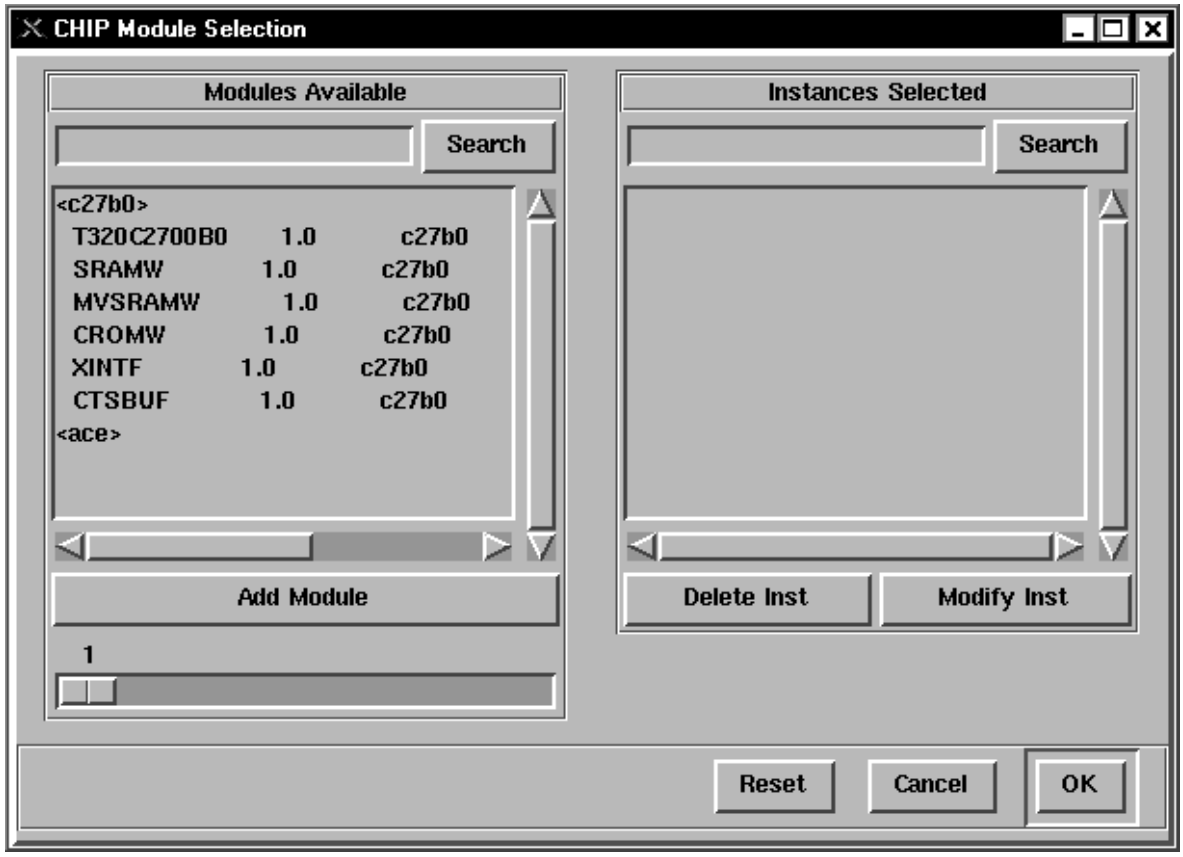
1) From the Edit MRS menu on the DSPnetGEN History window, choose Select Modules to display the CHIP Module Selection dialog box.



2) In the CHIP Module Selection dialog box, double click on the <c27b0> option listed in the Modules Available panel on the left-hand side. An expanded list of modules available in the <c27b0> module library appears.

Note:

If the <c27b0> option does not appear in the Modules Available panel, you must install the module library in your DSPnetGEN directory before you proceed any further. Contact Customer Support for details (see *Preface*).



3) From the list of available modules, select the T320C2700B0 core.

4) Click on the Add Module button to add the selected module to your design.

   An instance of the T320C2700B0 cDSP core appears in the right-hand pane of the CHIP Module Selection dialog box under the Instances Selected heading.

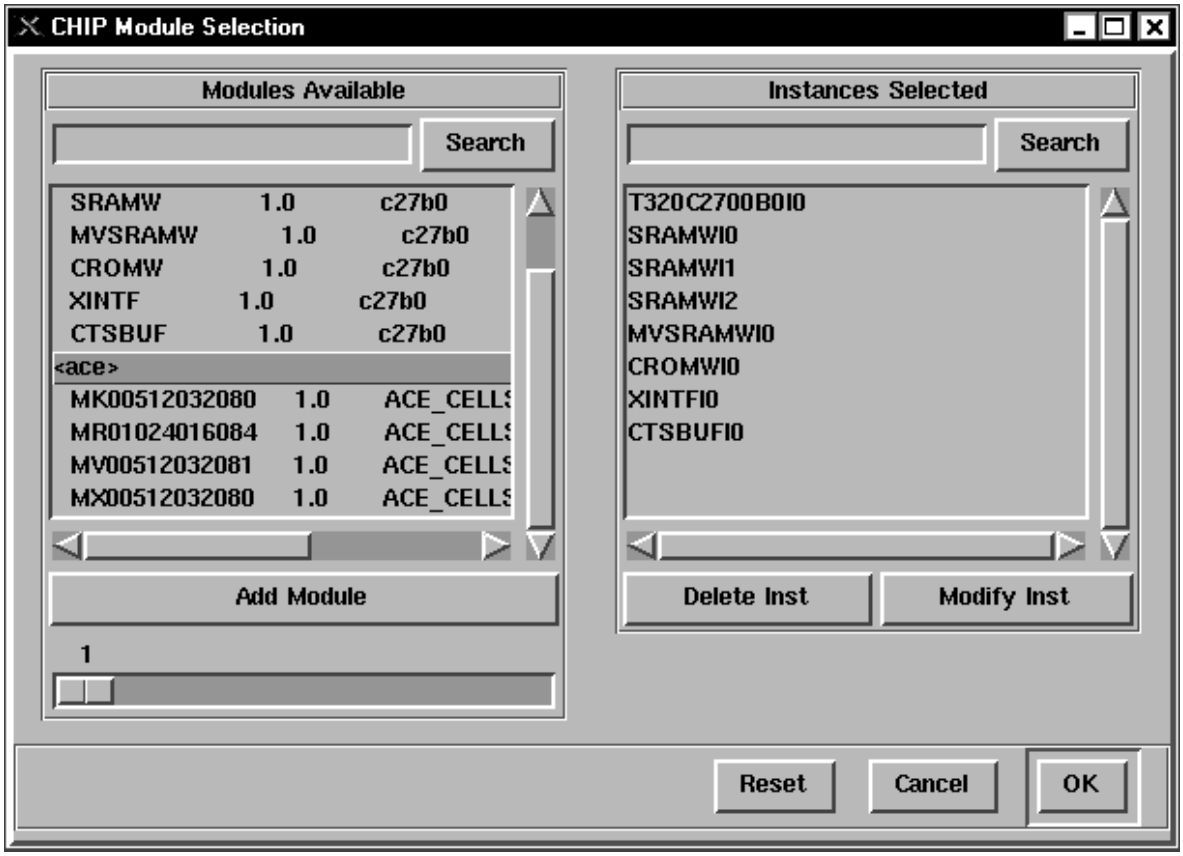
5) Select the remaining <c27b0> modules and click on the Add Module button to add the selected modules to your design. For the example design, these modules are specified in the module inventory list in section C.2 on page C-3 (also see the following Note).

Note:

The remaining <c27b0> modules that must be added to the example design are SRAMW (3), MVSRAMW, CROMW, XINTF, and CTSBUF.

To add additional instances of a module to your design, select the module and click the Add Module button multiple times.

6) To add <ace> memories to your design, double-click on the <ace> option in the Modules Available panel on the left-hand side of the CHIP Module Selection dialog box.



7) From the <ace> module library, select the <ace> memory modules to add to your design. For the example design, these modules are specified in the module inventory list in section C.2 on page C-3 (also see the following Note).

Note:

The <ace> memory modules that must be added to the example design are MK00512032080 (2), MR01024016084 (2), MV00512032081, and MX01024032080.

8) Click the Add Module button to add the selected <ace> memories to your design.

   Click on the Add Module button multiple times to add multiple instances of a module to your design.

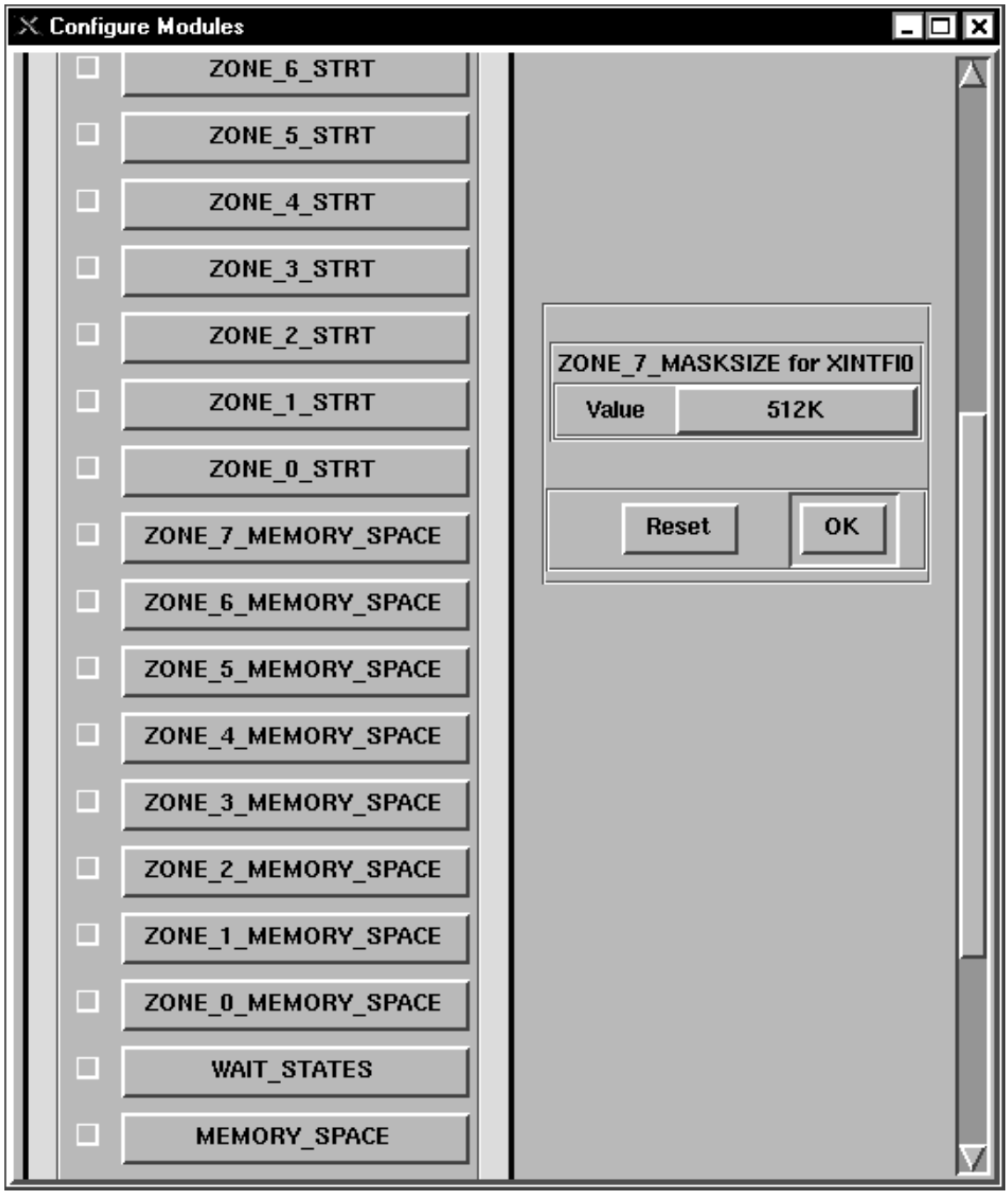
9) Click OK to store the instance information to the current MRS.

C.5 Configuring Modules

The modules that you entered into the MRS during section C.4, *Selecting Modules*, have certain parameters. This step allows you to customize these parameters to fit design-specific requirements.

To enter these parameters into DSPnetGEN, follow these steps:

1) From the Edit MRS menu on the DSPnetGEN History window, select Configure Modules to display the Configure Modules dialog box .



2) Click on the MEMORY_SPACE button to display the memory space information panel on the right-hand side of the Configure Modules dialog box.

Note:

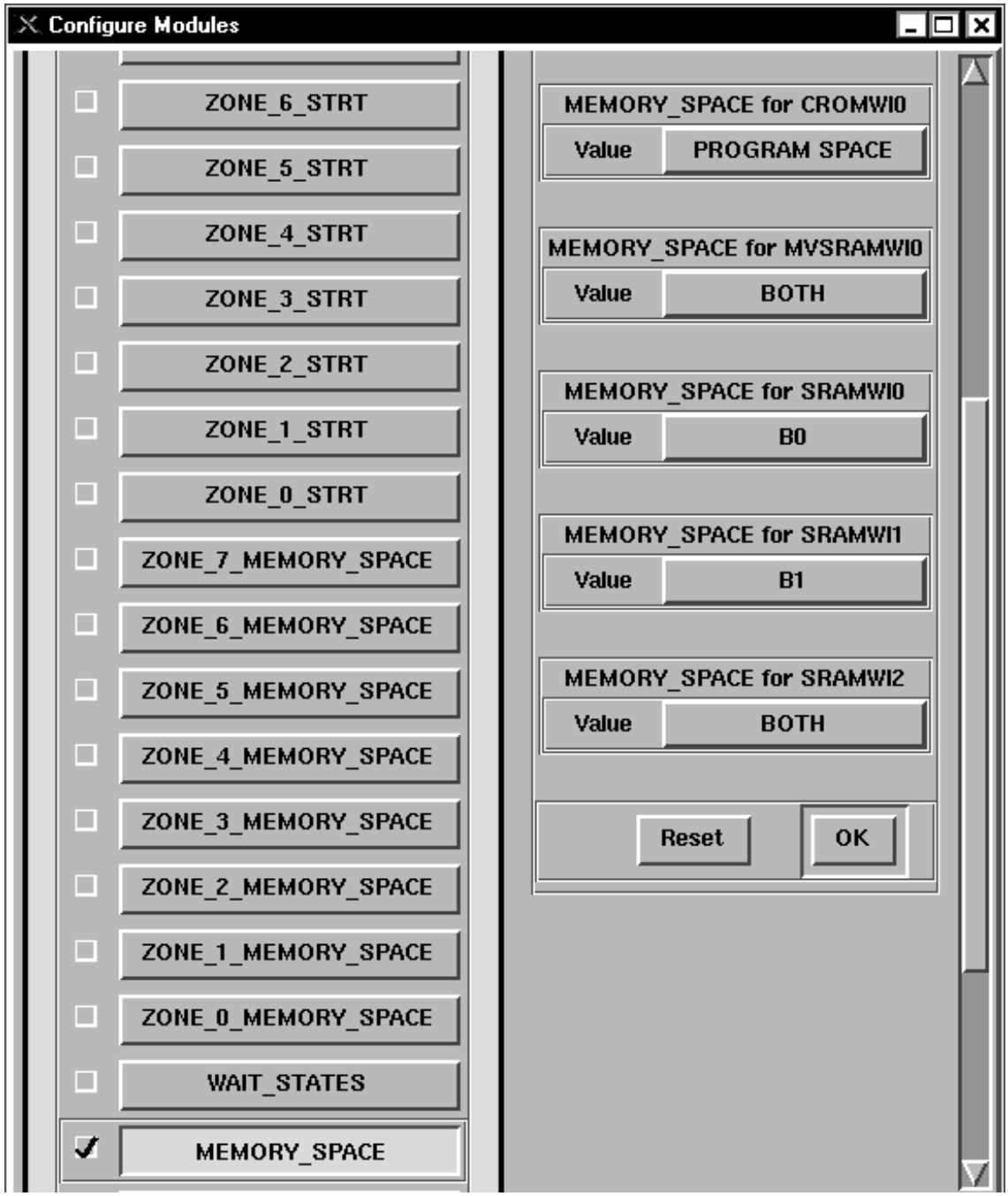
This parameter maps the selected memories to program and/or data spaces.

3) From the MEMORY_SPACE Value drop-down menu, select a value for each of the memory spaces. For the example design, select these values for each memory space :

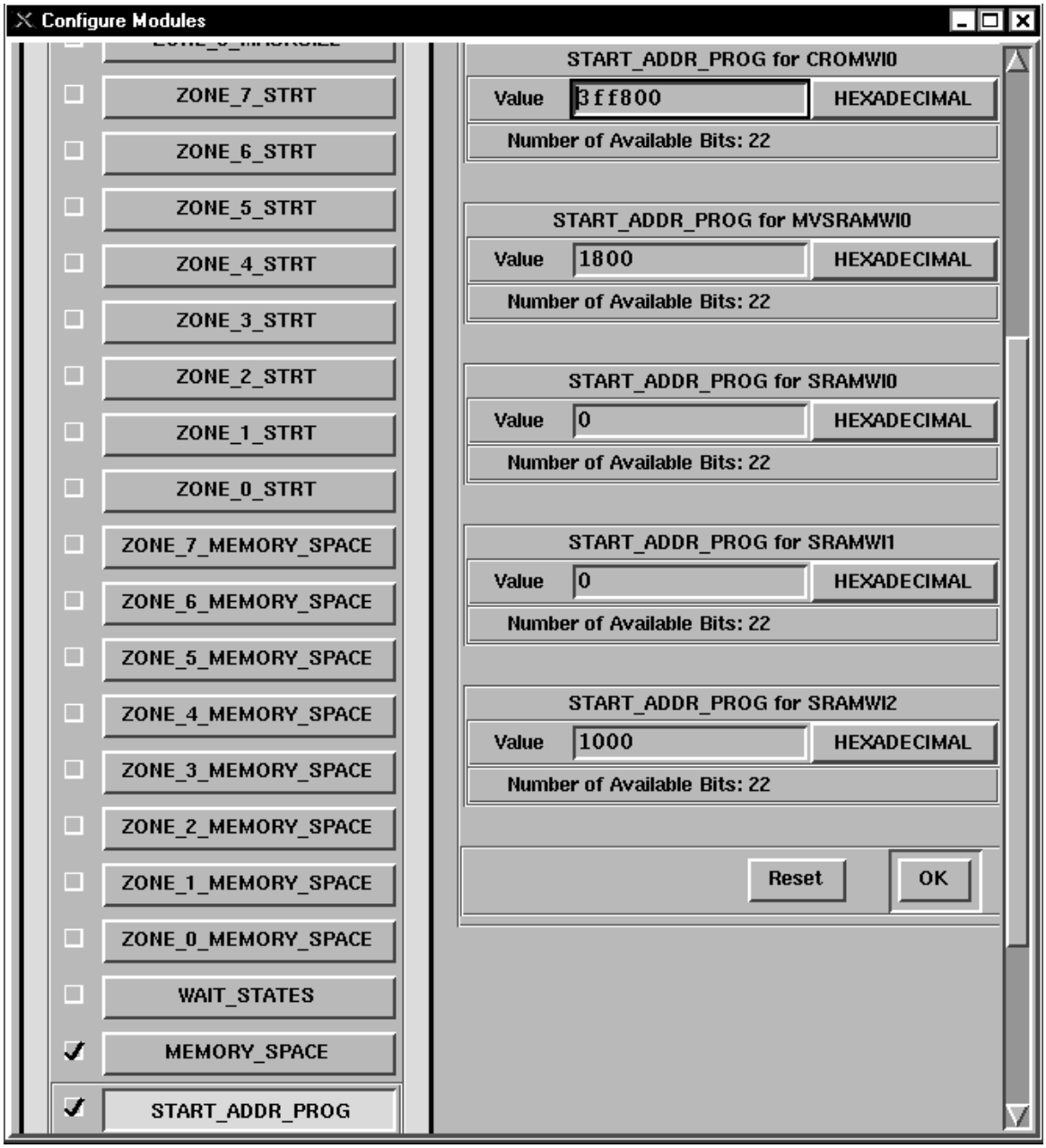
- ❑ For the CROMWI0 memory space, select PROGRAM SPACE.
- ❑ For the MVSRAMWI0 memory space, select BOTH.
- ❑ For the SRAMWI0 memory space, select B0.
- ❑ For the SRAMWI1 memory space, select B1.
- ❑ For the SRAMWI2 memory space, select BOTH.

4) Click OK to place a checkmark next to the MEMORY_SPACE parameter.

The checkmark verifies that you accept the values for a parameter.



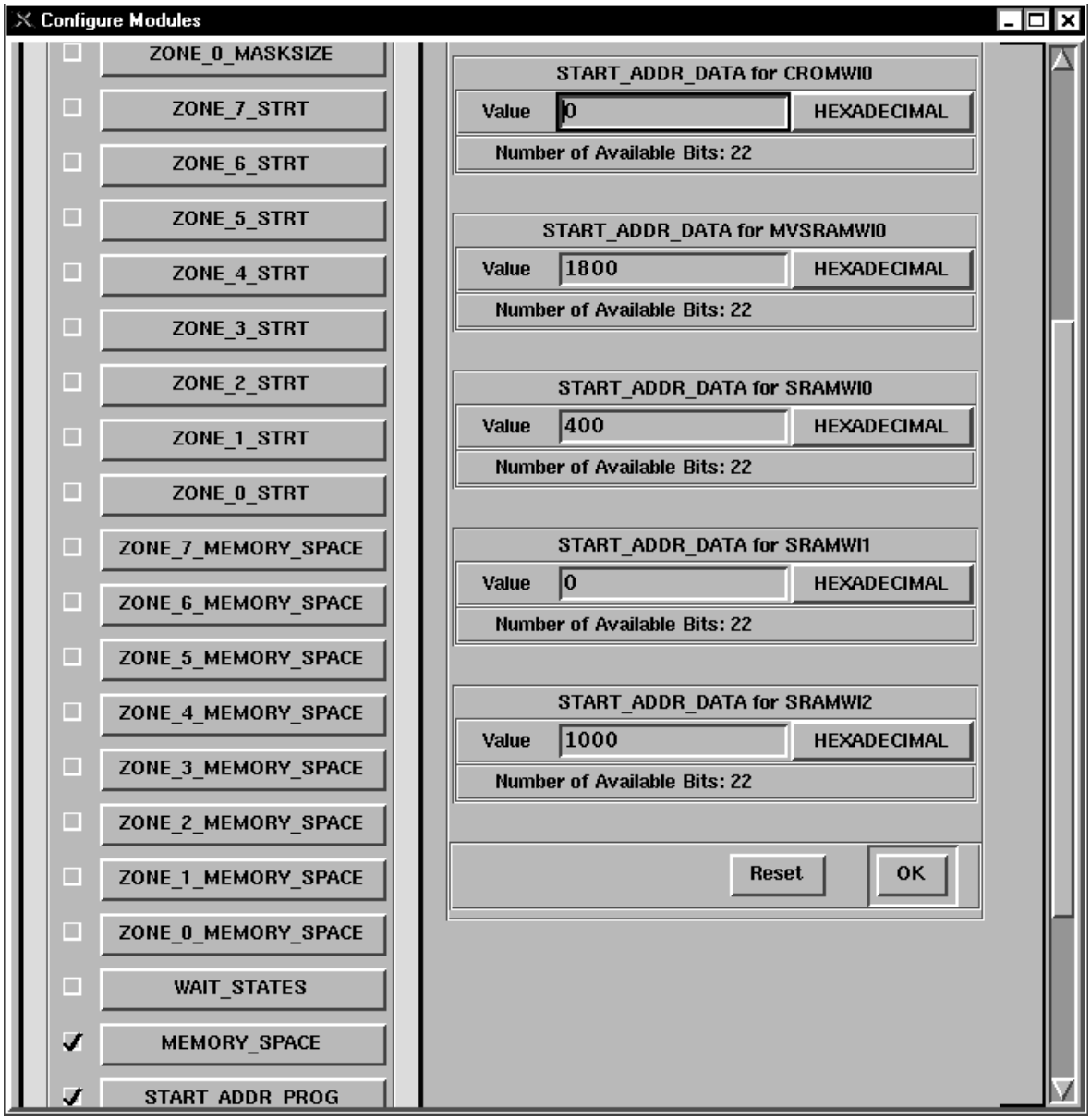
5) Click on the START_ADDR_PROG button to display the start address program information panel on the right-hand side of the Configure Modules dialog box.



6) Enter a hexadecimal value in each START_ADDR_PROG field and click OK. For the example design, enter these values for each start address program:

- ❑ For the CROMWI0 start address program, enter a hexadecimal value of 3ff800.
- ❑ For the MVSRAMWI0 start address program, enter a hexadecimal value of 1800.
- ❑ For the SRAMWI0 start address program, enter 0.
- ❑ For the SRAMWI1 start address program, enter 0.
- ❑ For the SRAMWI2 start address program, enter a hexadecimal value of 1000.

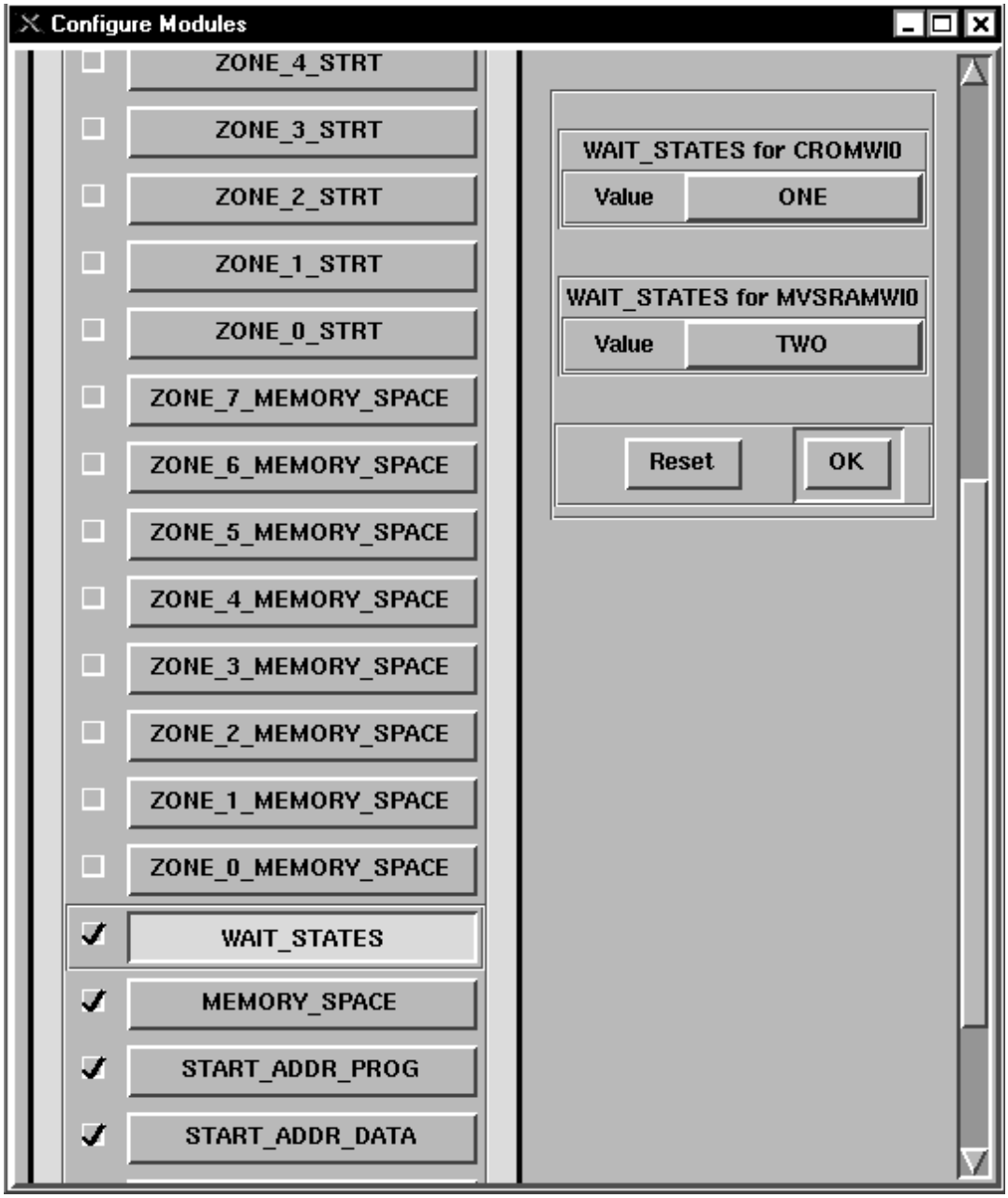
7) Click on the START_ADDR_DATA button to display the start address data information panel on the right-hand side of the Configure Modules dialog box.



8) Enter a hexadecimal value in each START_ADDR_DATA field and click OK. For the example design, enter these values for each start address data:

- ❑ For the CROMWI0 start address data, enter a hexadecimal value of 3ff800.
- ❑ For the MVSRAMWI1 start address data, enter a hexadecimal value of 1800.
- ❑ For the SRAMWI0 start address data, enter a hexadecimal value of 400.
- ❑ For the SRAMWI1 start address data, enter 0.
- ❑ For the SRAMWI2 start address data, enter a hexadecimal value of 1000.

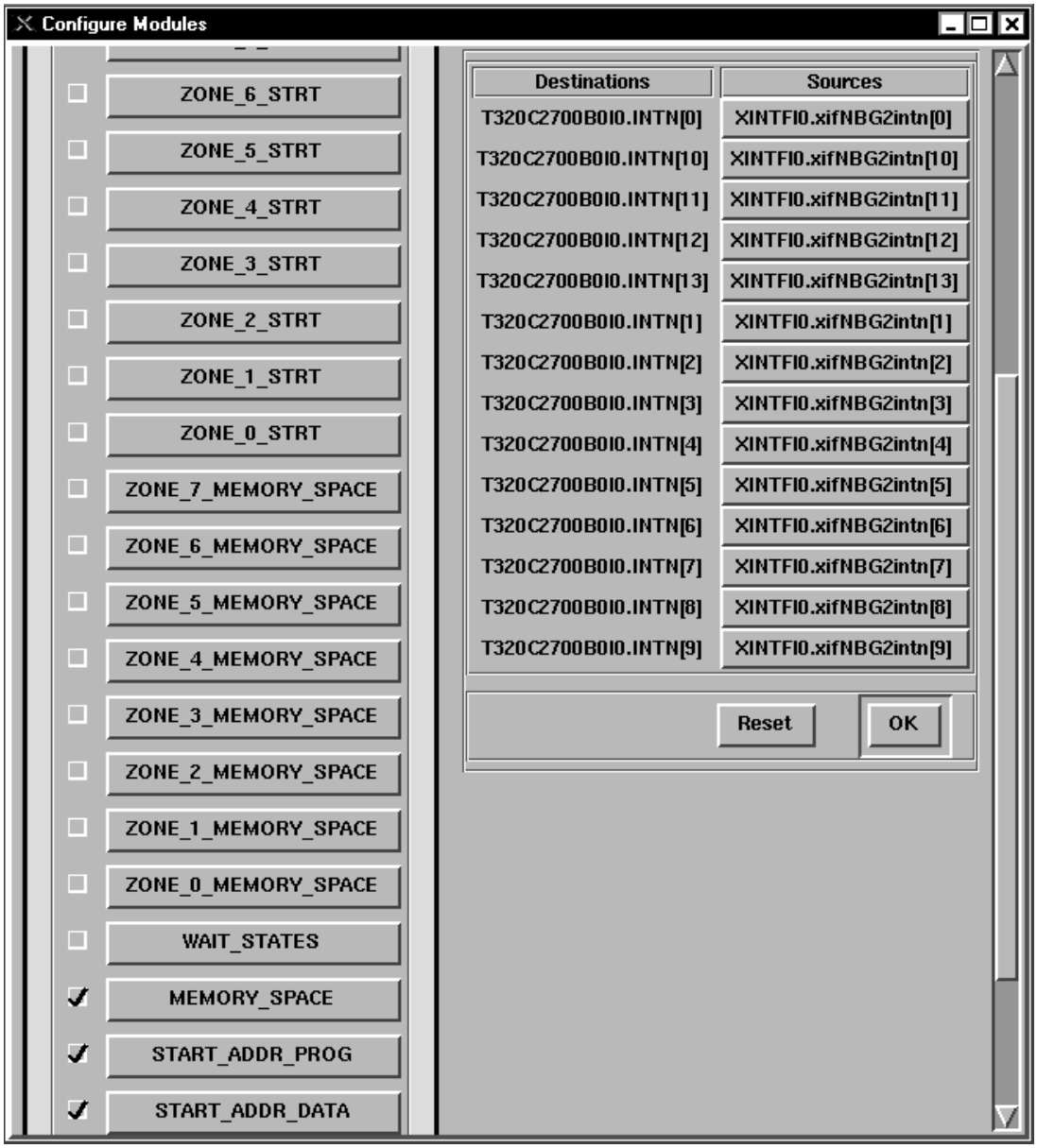
9) Click on the WAIT_STATES button to display the wait states information panel on the right-hand side of the Configure Modules dialog box .



10) From the WAIT_STATES Value drop-down menu, select a value for each of the wait states. For the example design, select these values for each wait state:

- ❑ For the CROMWI0 wait state, select ONE.
- ❑ For the MVSRAMWI0 wait state, select TWO.

11) Click on the INTERRUPTS button to display the interrupts information panel on the right-hand side of the Configure Modules dialog box.

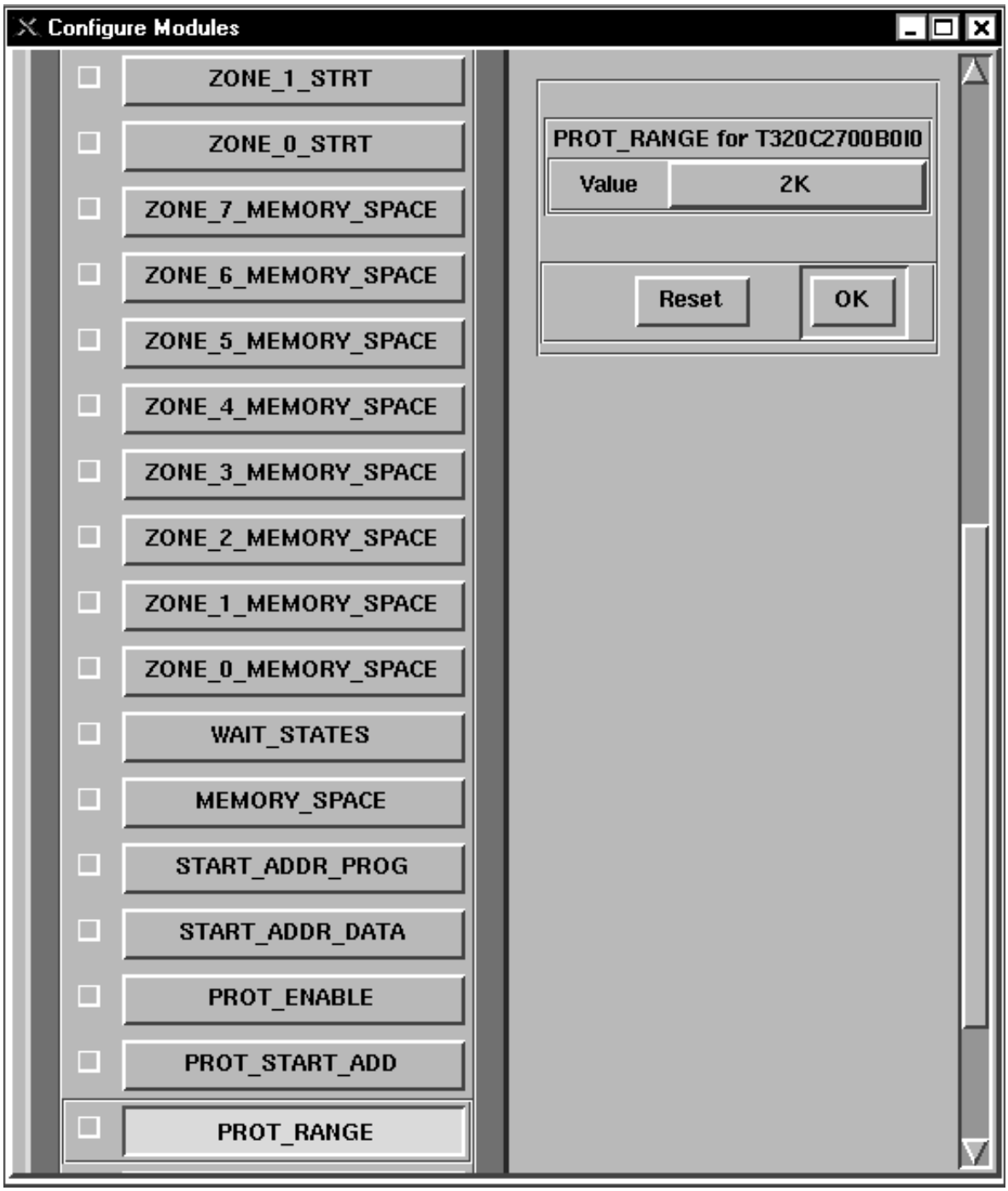


12) From the Sources drop-down menu, select a source for each of the core interrupt signals in the Destination column and click OK. Select the source according to the following criteria:

- ❑ Select HI for any unused interrupts.
- ❑ Select PROP to propagate an interrupt signal to the DSPnetGEN net-list output port map for use at a higher design hierarchy level.
- ❑ Select a XINTF bridge to connect the interrupt signal to the XINTF.

For the example design, select a XINTF bridge that corresponds to its T320C2700B0I0.INTN interrupt signal in the Destination column. See Table C–2 on page C-4 for a list of core interrupt signals and their sources.

13) Click on the PROT_RANGE button to display the protection range information panel on the right-hand side of the Configure Modules dialog box.

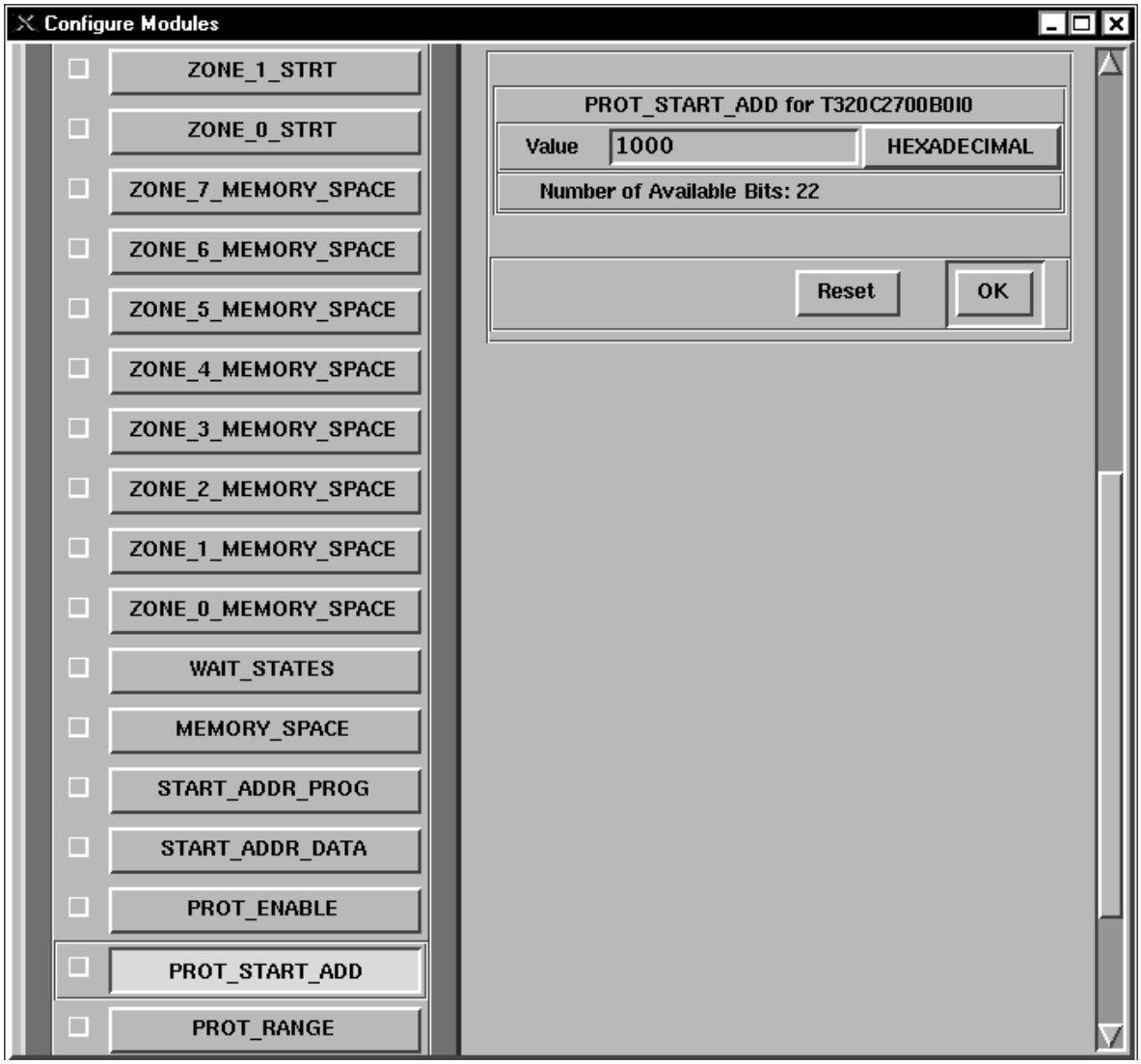


Note:

This parameter sets the block size that enables protection from the starting address.

14) From the PROT_RANGE Value drop-down menu, select a value and click OK. For the example design, select a value of 2K.

15) Click on the PROT_START_ADD button to display the protection start address information panel on the right-hand side of the Configure Modules dialog box.

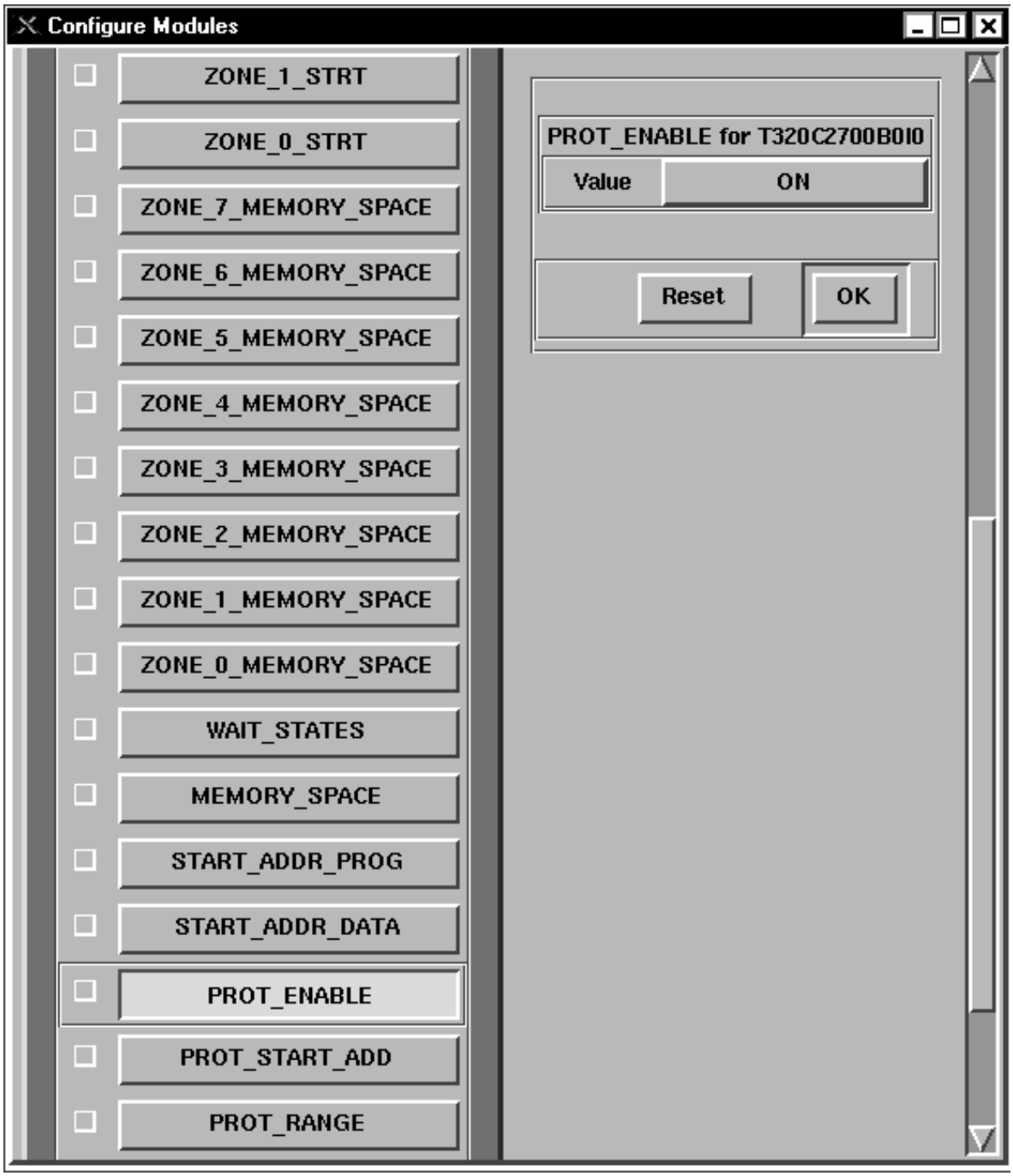


Note:

This parameter specifies the memory start address that enables protection for a write followed by read operations. The PROT_START_ADD value must be a multiple of the PROT_RANGE value.

16) In the PROT_START_ADD Value field, enter a hexadecimal value and click OK. For the example design, enter a hexadecimal value of 1000.

17) Click on the PROT_ENABLE button to display the protection enable information panel on the right-hand side of the Configure Modules dialog box.

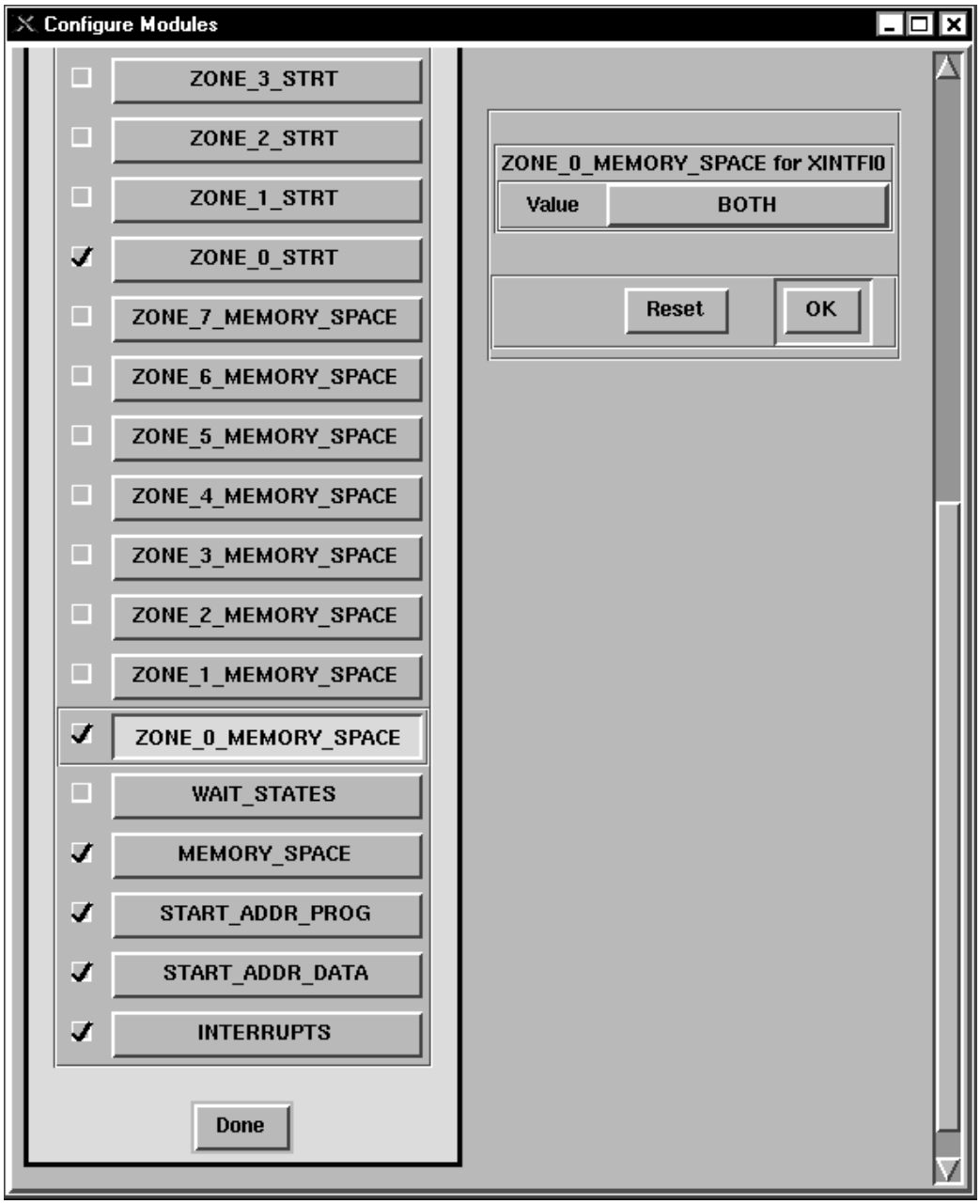


18) From the PROT_ENABLE Value drop-down menu, select a value and click OK. Select the PROT_ENABLE value according to the following criteria:

- ❑ Select ON if the ENPROT signal for the ’C2700B0 core is tied high.
- ❑ Select OFF if the ENPROT signal for the ’C2700B0 core is tied low.

For the example design, select ON for the PROT_ENABLE value.

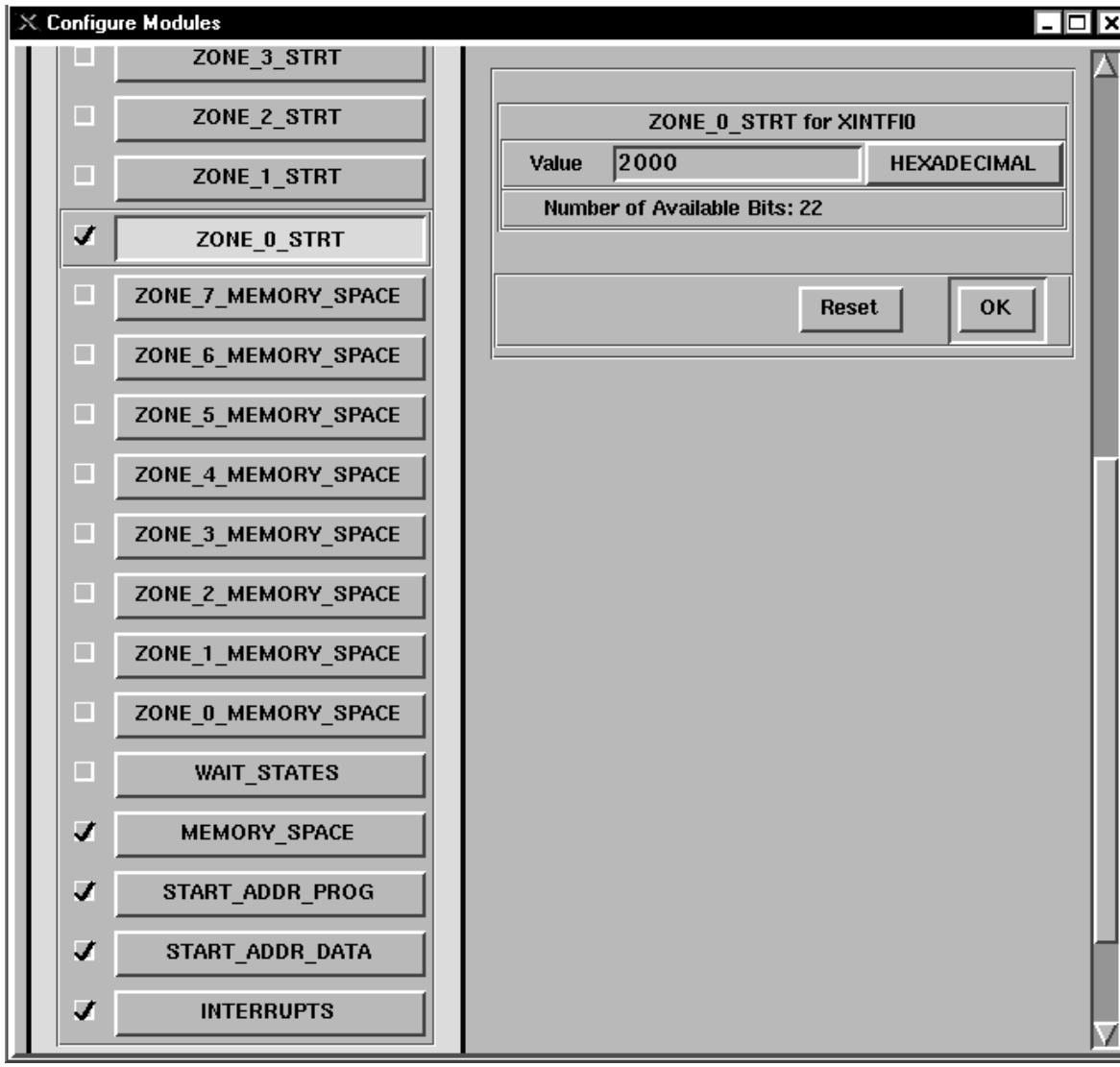
19) Click on the ZONE_0_MEMORY_SPACE button to display the memory space information panel on the right-hand side of the Configure Modules dialog box.



20) From the ZONE_0_MEMORY_SPACE Value drop-down menu on the right-hand side, select a value and click OK. For the example design, select BOTH for the ZONE_0_MEMORY_SPACE value.

21) From the Value drop-down menus of the remaining memory space zones (MEMORY_SPACE zones 1 to 7), select a value and click OK. For the example design, use the default values. For a list of default values, see Table C–3 on page C-5.

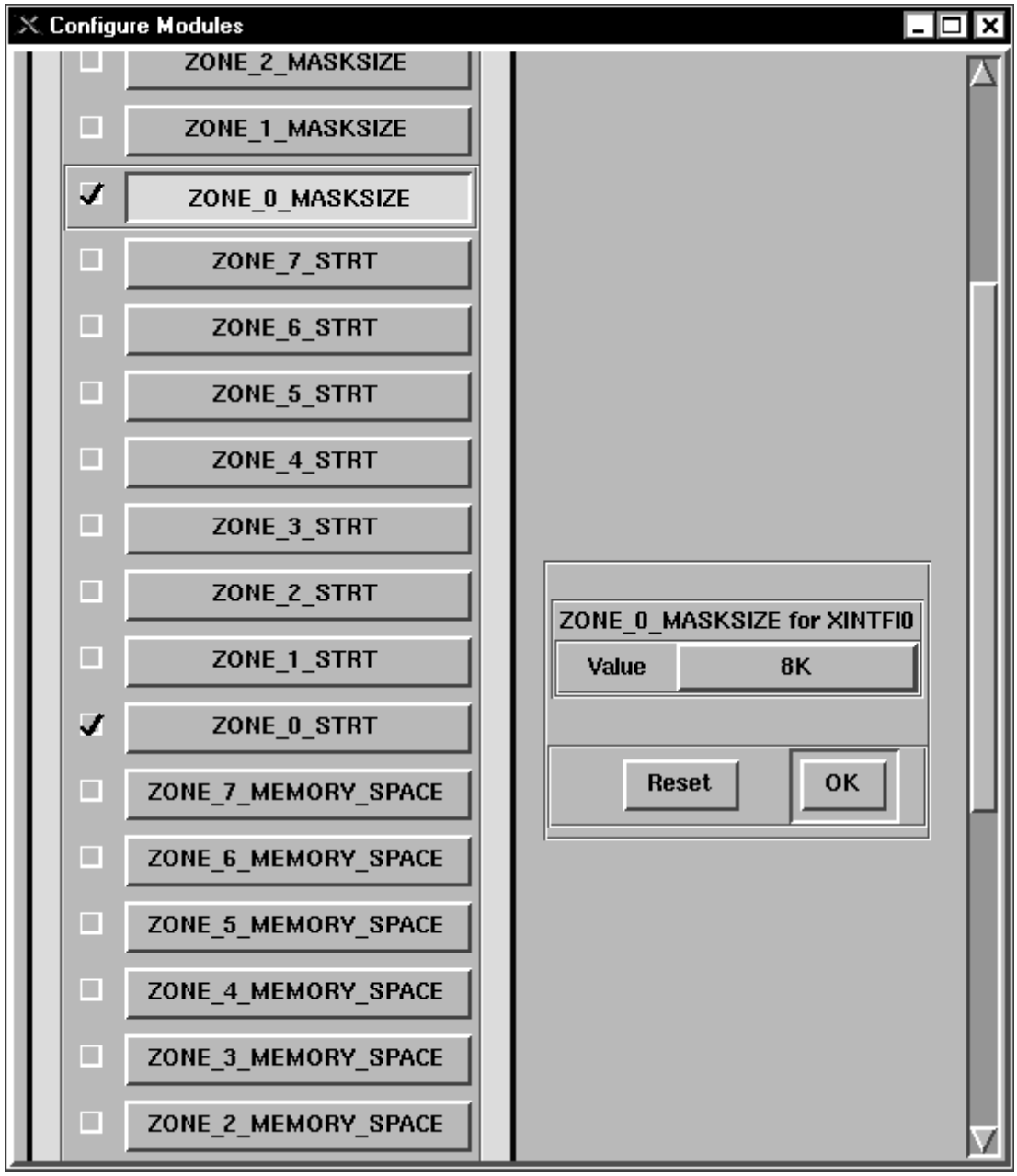
22) Click on the ZONE_0_STRT button in the left-hand pane of the Configure Modules dialog box.



23) Enter a hexadecimal value in the ZONE_0_STRT Value field on the right-hand side and click OK. For the example design, enter a hexadecimal value of 2000.

24) For the remaining start zones (STRT zones 1 to 7), enter a hexadecimal value in the Value field on the right-hand side and click OK. For the example design, use the default values. For a list of default values, see Table C–3 on page C-5.

25) Click on the ZONE_0_MASKSIZE button in the left-hand pane of the Configure Modules dialog box.



26) From the ZONE_0_MASKSIZE Value drop-down menu on the right-hand side, select a value and click OK. For the example design, select a ZONE_0_MASKSIZE value of 8K.

27) From the Value drop-down menus of the remaining masksize zones (MASKSIZE zones 1 to 7), select a value and click OK. For the example design, use the default values. For a list of default values, see Table C–3 on page C-5.

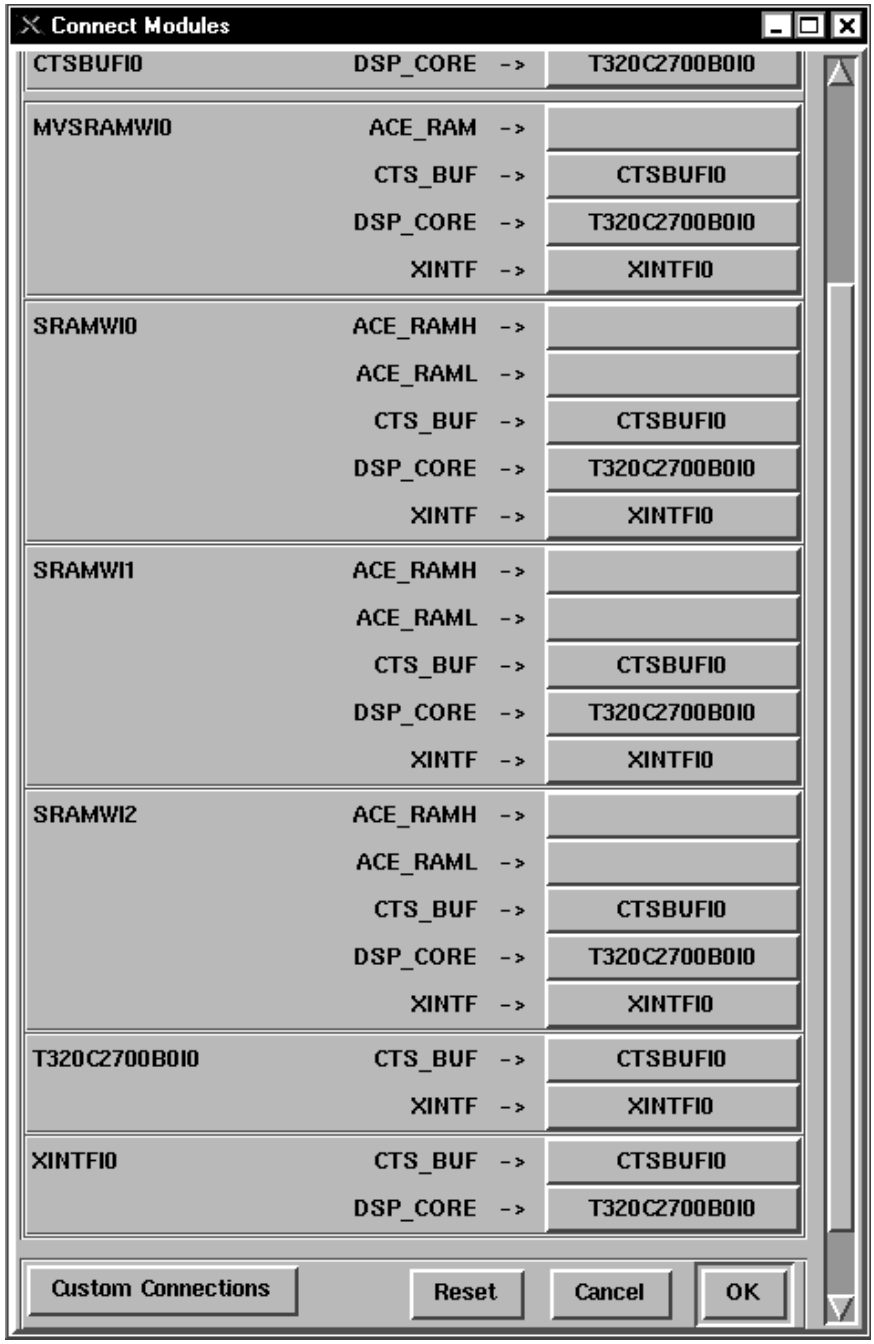
28) Click on the Done button to display a module parameter report in the DSPnetGEN History window.

C.6 Determining Connections

This step allows you to determine the nets for the design. In most cases, the connection engine can automatically identify module mappings and determine corresponding nets for the design. However, in cases where multiple instances of a module exist, the connection engine may be unable to resolve module mappings automatically. In such cases, you must provide the desired mapping between the modules.

To specify the mapping between modules, follow these steps:

1) From the Edit MRS menu, select Connect Modules to display the Connect Modules dialog box.



2) From the MVSRAMWI0 ACE_RAM drop-down menu in the Connect Modules dialog box, select the desired ACE memory core. For the example device, select MV005120320 81I0.

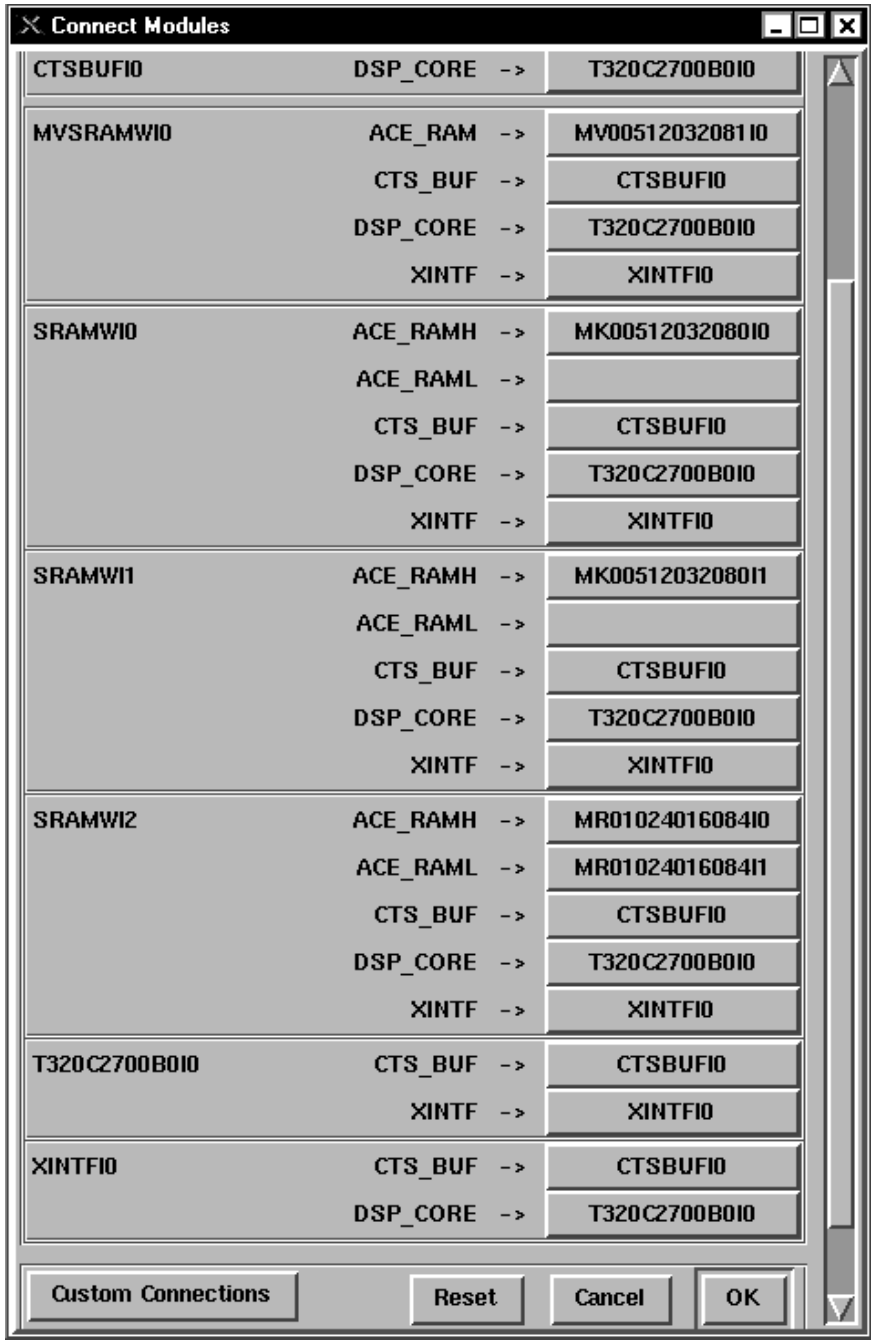
3) From the ACE_RAMH and ACE_RAML drop-down menus for SRAMWI0, SRAMWI1, and SRAMWI2 memory wrappers, select the desired ACE memory cores. For the example device, select the ACE memory cores as follows:

   - ❑ From the SRAMWI0 ACE_RAMH drop-down menu, select the MK00512032080I0 memory core.
   - ❑ From the SRAMWI1 ACE_RAMH drop-down menu, select the MK00512032080I1 memory core.
   - ❑ From the SRAMWI2 ACE_RAMH drop-down menu, select the MR0102401608 4I0 memory core.
   - ❑ From the SRAMWI2 ACE_RAML drop-down menu, select the MR0102401608 4I1 ACE_RAML memory core.

Note:

You do not need to determine connections for the remaining module instances. DSPnetGEN is able to automatically determine the required instance mapping for the remaining device modules.

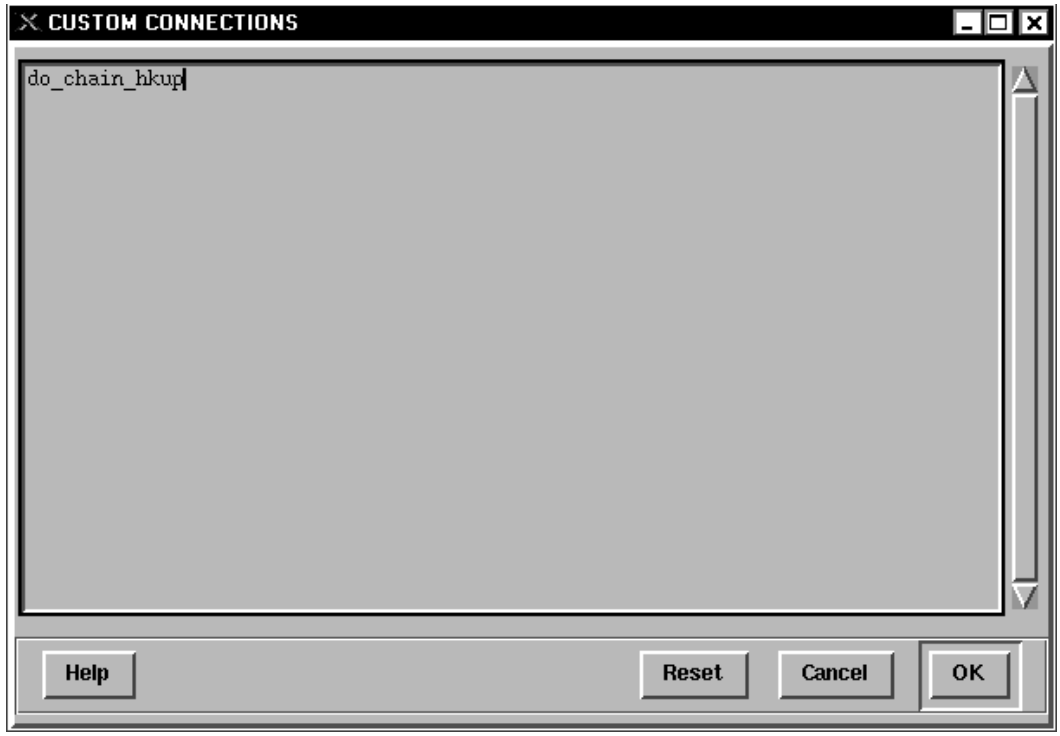
The preceding specifications are illustrated as follows:



4) Click on the Custom Connections button to display the CUSTOM CONNECTIONS window.

Note:

You can use the CUSTOM CONNECTIONS window to override the connection rules found in a module library with actions such as adding terminals or adding/deleting individual connections. You can also use it as a means to utilize additional library-specific functionality. Edit the GNF commands in the CUSTOM CONNECTIONS window to customize your design.



5) In the CUSTOM CONNECTIONS text entry window, enter the command `do_chain_hkup` to enable the function that automates PRDY, DRDY, and scan chain connections and click OK to return to the Connect Modules dialog box.

6) In the Connect Modules dialog box, click OK to generate detailed netlist information in the DSPnetGEN History window. DSPnetGEN updates the current MRS with the results.

C.7 Generating a Structural Model

To produce a netlist for the design, follow these steps:

1) From the Tools menu on the DSPnetGEN History window, select Write VHDL to write a VHDL netlist to your design. You can also write a Verilog netlist using the same basic process.

2) Enter filename.vhd for the VHDL filename. For our example design, enter `example_design.vhd`.

3) Specify the template if you want to use a template other than the default one in your design. For the example design, leave the Template field blank.

4) Click OK to display the requested netlist in the DSPnetGEN History window.

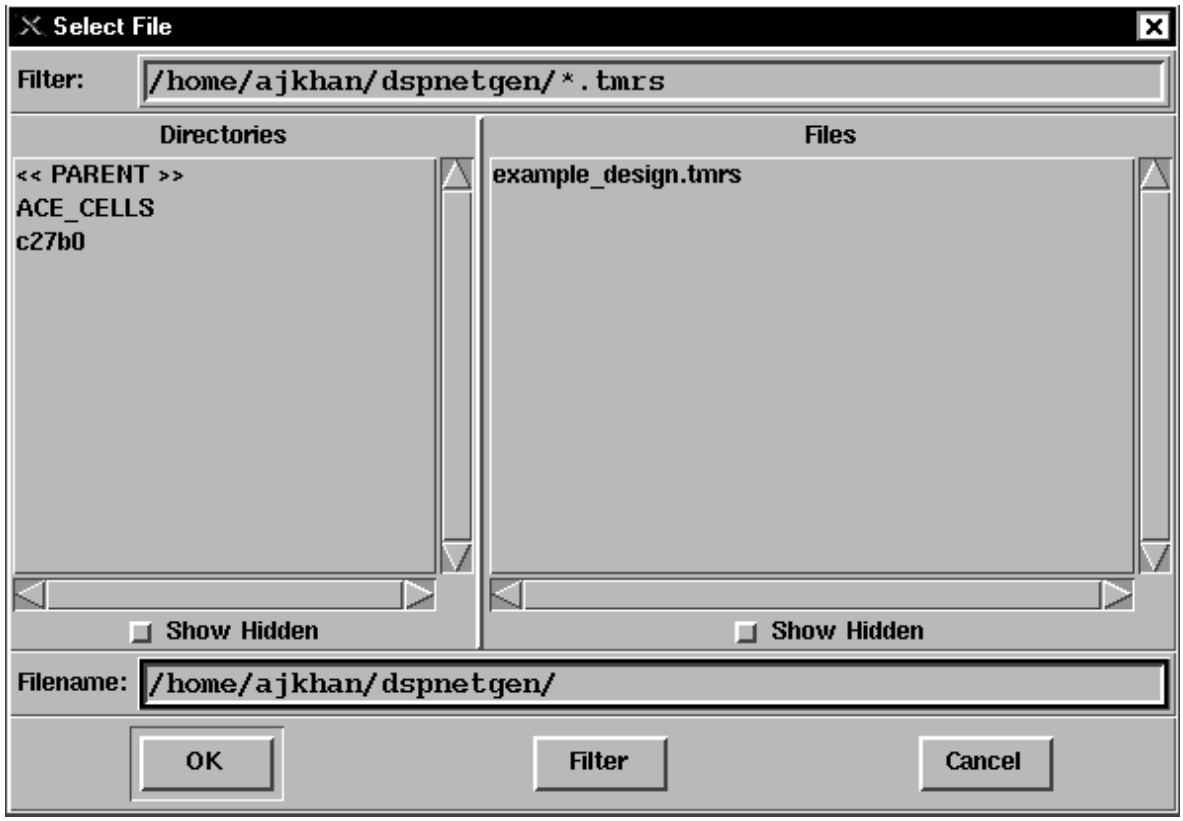
C.8 Ending DSPnetGEN

To end DSPnetGEN, you must first save your current MRS to a file and then exit DSPnetGEN.

C.8.1 Saving Machine Readable Specification (MRS) to a File

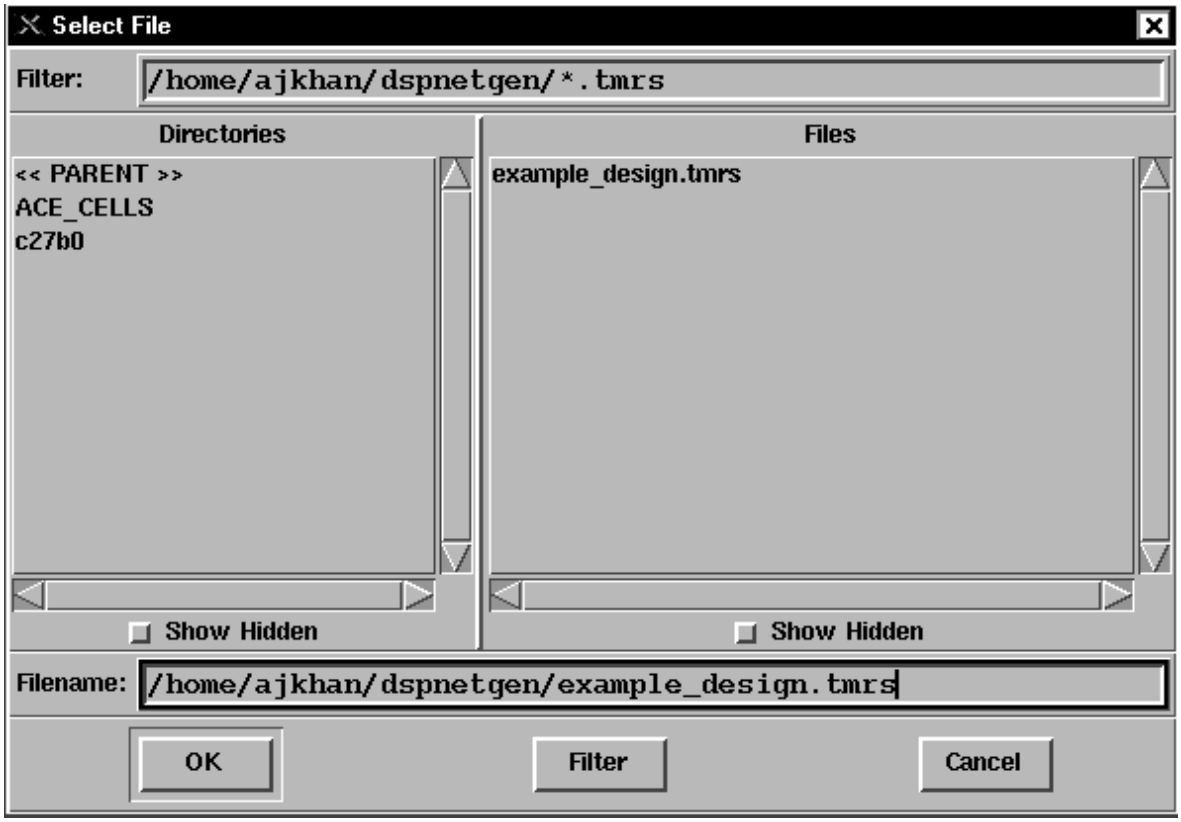
To save your MRS to a file, follow these steps:

1) From the File Menu on the DSPnetGEN History window, select Save As... to display the Select File dialog box.



2) In the Filename field, enter the name of the file in which to save your MRS.

For the example design, enter `example_design.tmrs` for the filename.



3) Click OK to save the MRS and to display the save operation status in the DSPnetGEN History window.

C.8.2 Exiting DSPnetGEN

To exit from DSPnetGEN, follow these steps:

1) From the File menu on the DSPnetGEN History window, select Exit.

   You can exit from DSPnetGEN only if the system saved changes made to the MRS.

If DSPnetGEN did not save changes, selecting Exit will display the MRS MODIFIED warning dialog box.

2) Click Save to save the MRS to a file before exiting DSPnetGEN.
Click Discard to exit DSPnetGEN without saving the MRS to a file.
Click Cancel to abort the exit command.

Index

A

B

C

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z