

TMS320C5x Simulator Getting Started Guide

Literature Number: SPRU124C
Reprinted November 1997



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software 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, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

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

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI 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.

Preface

Read This First

About This Manual

This manual describes how to install the TMS320C5x simulator and the C source debugger on PC™ systems running MS-DOS™ or PC-DOS™; on a SPARCstation running SunOS™ or Solaris™; and on a HP9000 series 700™ PA-RISC™ system running HP-UX™.

This manual also contains release notes for the TMS320C5x simulator.

Notational Conventions

This document uses the following conventions.

- Program listings, program examples, and interactive displays are shown in a special typeface similar to a typewriter's. Examples use a **bold version** of the special typeface for emphasis; interactive displays use a **bold version** of the special typeface to distinguish commands that you enter from items that the system displays (such as prompts, command output, error messages, etc.).

Here is a sample program listing:

```
0011 0005 0001      .field  1, 2
0012 0005 0003      .field  3, 4
0013 0005 0006      .field  6, 3
0014 0006           .even
```

Here is an example of a system prompt and a command that you might enter:

```
C:  csr -a /user/ti/simuboard/utilities
```

- In syntax descriptions, the instruction, command, or directive is in a **bold typeface** font and parameters are in an *italic typeface*. Portions of a syntax that are in **bold** should be entered as shown; portions of a syntax that are in *italics* describe the type of information that should be entered. Here is an example of a directive syntax:

.asect *"section name", address*

.asect is the directive. This directive has two parameters, indicated by *section name* and *address*. When you use .asect, the first parameter must be an actual section name, enclosed in double quotes; the second parameter must be an address.

- Square brackets ([and]) identify an optional parameter. If you use an optional parameter, you specify the information within the brackets; you don't enter the brackets themselves. Here's an example of an instruction that has an optional parameter:

LALK *16-bit constant [, shift]*

The LALK instruction has two parameters. The first parameter, *16-bit constant*, is required. The second parameter, *shift*, is optional. As this syntax shows, if you use the optional second parameter, you must precede it with a comma.

Square brackets are also used as part of the pathname specification for VMS pathnames; in this case, the brackets are actually part of the pathname (they are not optional).

- Braces ({ and }) indicate a list. The symbol | (read as *or*) separates items within the list. Here's an example of a list:

{ * | *+ | *- }

This provides three choices: *, *+, or *-.

Unless the list is enclosed in square brackets, you must choose one item from the list.

- Some directives can have a varying number of parameters. For example, the .byte directive can have up to 100 parameters. The syntax for this directive is:

.byte *value₁ [, ... , value_n]*

This syntax shows that .byte must have at least one value parameter, but you have the option of supplying additional value parameters, separated by commas.

Related Documentation From Texas Instruments

The following books describe the TMS320C5x 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.

TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User's Guide (literature number SPRU018) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the 'C1x, 'C2x, 'C2xx, and 'C5x generations of devices.

TMS320C2x/C2xx/C5x Optimizing C Compiler User's Guide (literature number SPRU024) describes the 'C2x/'C2xx/'C5x C compiler. This C compiler accepts ANSI standard C source code and produces TMS320 assembly language source code for the 'C2x, 'C2xx, and 'C5x generations of devices.

TMS320C5x C Source Debugger User's Guide (literature number SPRU055) tells you how to invoke the 'C5x emulator, EVM, 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, and includes a tutorial that introduces basic debugger functionality.

TMS320C5x User's Guide (literature number SPRU056) describes the 'C5x 16-bit, fixed-point, general-purpose digital signal processors. Covered are its architecture, internal register structure, instruction set, pipeline, specifications, DMA, I/O ports, and on-chip peripherals.

If You Need Assistance . . .

If you want to . . .	Contact Texas Instruments at . . .
Visit TI online, including TI&ME™, your own customized web page	World Wide Web: http://www.ti.com
Receive general information or assistance	World Wide Web: http://www.ti.com/sc/docs/pic/home.htm North America and South America (English): (214) 644-5580 Europe, Middle East, and Africa: Dutch: 33-130-70-1166 English: 33-130-70-1165 French: 33-130-70-1164 Italian: 33-130-70-1167 German: 33-130-70-1168 Japan (Japanese or English): Domestic toll-free: 0120-81-0026 International: 81-3-3457-0972 or 81-3-3457-0976 Korea (Korean or English): 82-2-551-2804 Taiwan (Chinese or English): 886-2-3771450
Ask questions about Digital Signal Processor (DSP) product operation or report suspected problems	(713) 274-2320 Fax: (713) 274-2324 Fax Europe: +33-1-3070-1032 Email: 4389750@mcimail.com World Wide Web: http://www.ti.com/sc/docs/dsp/expsys.htm BBS North America: (713) 274-2323 8-N-1 BBS Europe: +44-2-3422-3248 320 BBS Online: ftp://ti.com/mirrors/tms320bbs (192.94.94.33)
Request tool updates	Software: (214) 638-0333 Fax: (214) 638-7742 Hardware: (713) 274-2285
Order Texas Instruments documentation (a literature number is required)	(800) 477-8924
Make suggestions about or report mistakes in documentation (please mention the full title of the book, the literature number, and the publication date from the spine or front cover)	Email: comments@books.sc.ti.com Texas Instruments Incorporated Technical Publications Manager, MS 702 P.O. Box 1443 Houston, Texas 77251-1443

Trademarks

HP 9000 Series 700, HP-UX, and PA-RISC are trademarks of Hewlett-Packard Company.

MS-DOS and Windows are registered trademarks of Microsoft Corp.

OpenWindows, Solaris, and SunOS are trademarks of Sun Microsystems, Inc.

PC and PC-DOS are trademarks of International Business Machines Corp.

SPARC is a trademark of SPARC International, Inc.

SPARCstation is licensed exclusively to Sun Microsystems, Inc.

TI&ME is a trademark of Texas Instruments Incorporated.

X Window System is a trademark of the Massachusetts Institute of Technology.

Contents

1	Installing the Simulator and C Source Debugger With DOS	1-1
	<i>Lists the hardware and software you need to install the simulator and C source debugger; provides installation instructions for PC systems running MS-DOS or PC-DOS.</i>	
1.1	What You Need	1-2
	Hardware checklist	1-2
	Software checklist	1-3
1.2	Step 1: Installing the Simulator and Debugger Software	1-4
1.3	Step 2: Setting Up the Debugger Environment	1-5
	Defining an initialization batch file	1-6
	Changing the autoexec.bat file	1-7
1.4	Step 3: Verifying the Installation	1-8
1.5	Using the Debugger With Windows	1-9
2	Installing the Simulator and C Source Debugger With SunOS	2-1
	<i>Lists the hardware and software you need to install the simulator and C source debugger; provides installation instructions for SPARCstations running SunOS or Solaris.</i>	
2.1	What You Need	2-2
	Hardware checklist	2-2
	Software checklist	2-3
2.2	Step 1: Installing the Simulator and Debugger Software	2-4
	Mounting the CD-ROM	2-4
	Copying the files	2-5
	Unmounting the CD-ROM	2-5
2.3	Step 2: Setting Up the Debugger Environment	2-6
	Reinitializing your shell	2-7
2.4	Step 3: Verifying the Installation	2-8
2.5	Using the Debugger With the X Window System	2-9
	Using the keyboard's special keys	2-9
	Changing the debugger font	2-10
	Color mappings on monochrome screens	2-10

3	Installing the Simulator and C Source Debugger With HP-UX	3-1
	<i>Lists the hardware and software you need to install the simulator and C source debugger; provides installation instructions for HP systems running HP-UX.</i>	
3.1	What You Need	3-2
	Hardware checklist	3-2
	Software checklist	3-3
3.2	Step 1: Installing the Simulator and Debugger Software	3-4
	Mounting the CD-ROM	3-4
	Copying the files	3-4
	Unmounting the CD-ROM	3-5
3.3	Step 2: Setting Up the Debugger Environment	3-6
	Reinitializing your shell	3-7
3.4	Step 3: Verifying the Installation	3-8
3.5	Using the Debugger With the X Window System	3-9
	Using the keyboard's special keys	3-9
	Changing the debugger font	3-10
	Color mappings on monochrome screens	3-10
4	Release Notes	4-1
	<i>Describes changes to the simulator version of the TMS320C5x C source debugger.</i>	
4.1	Changes to the Options Used to Invoke the C Source Debugger	4-2
4.2	Changes to Defining a Memory Map	4-2
4.3	Known Problem	4-2
5	Defining a Memory Map	5-1
	<i>Contains instructions for setting up a memory map that enables the debugger to correctly access target memory. Includes hints about using batch files, and tells you how to simulate I/O ports for use with the simulator version of the debugger.</i>	
5.1	The Memory Map: What It Is and Why You Must Define It	5-2
	Defining the memory map in a batch file	5-2
	Potential memory map problems	5-3
5.2	A Sample Memory Map	5-4
5.3	Identifying Usable Memory Ranges	5-6
	Memory mapping with the simulator (PC version)	5-7
5.4	Enabling Memory Mapping	5-8
5.5	Checking the Memory Map	5-9
5.6	Modifying the Memory Map During a Debugging Session	5-10
	Returning to the original memory map	5-11
5.7	Using Multiple Memory Maps for Multiple Target Systems	5-11
5.8	Simulating I/O Space (Simulator Only)	5-12
	Connecting an I/O port	5-12
	Disconnecting an I/O port	5-13

5.9	Simulating External Interrupts (Simulator Only)	5-14
	Setting up your input file	5-14
	Programming the simulator	5-16
5.10	Simulating Peripherals (Simulator Only)	5-18
5.11	Simulating Standard Serial Ports (Simulator Only)	5-19
	Setting up your transmit and receive operations	5-20
	Connecting input/output files	5-21
	Programming the simulator	5-21
5.12	Simulating Buffered Serial Ports (Simulator Only)	5-22
	Setting up your transmit and receive operations	5-23
	Connecting input/output files	5-24
	Programming the simulator	5-24
5.13	Simulating TDM Serial Ports (Simulator Only)	5-25
	Setting up your transmit and receive operations	5-26
	Connecting input/output files	5-27
	Programming the simulator	5-27

Figures

1-1	Sample Initialization Batch File	1-6
1-2	Sample autoexec.bat File	1-7
2-1	Sample Shell Configuration File for an X Window System	2-7
3-1	Sample Shell Configuration File for an X Window System	3-7
5-1	Memory Map Commands in the Sample Initialization Batch File for the EVM	5-4
5-2	Sample Memory Map for Use With a 'C5x EVM	5-5

Installing the Simulator and C Source Debugger With DOS

This chapter describes how to install the TMS320C5x simulator and the C source debugger on PC systems running MS-DOS or PC-DOS. You can also use the debugger with Windows™. When you complete the installation, turn to the *TMS320C5x C Source Debugger User's Guide*.

Topic	Page
1.1 What You Need	1-2
1.2 Step 1: Installing the Simulator and Debugger Software	1-4
1.3 Step 2: Setting Up the Debugger Environment	1-5
1.4 Step 3: Verifying the Installation	1-8
1.5 Using the Debugger With Windows	1-9

1.1 What You Need

To install the 'C5x C source debugger and simulator, you need the items in the following hardware and software checklists.

Hardware checklist

- Host** An IBM PC/AT or 100%-compatible ISA/EISA-based PC with a hard-disk system and a 1.2M floppy-disk drive; a 386 or higher is highly recommended
- Memory** Minimum of 640K bytes; in addition, if you are running under Windows, you need at least 256K bytes of extended memory
- Display** Monochrome or color monitor (color recommended)
- Optional hardware** A Microsoft-compatible mouse
- An EGA- or VGA-compatible graphics display card and a large monitor. The debugger has several options that allow you to change the overall size of the debugger display. If you have an EGA or VGA-compatible graphics card, you can take advantage of the larger screen sizes. The larger screen sizes are most effective when used with a large (17" or 19") monitor. (To use a larger screen size, you must invoke the debugger with an appropriate option. For more information about options, refer to the invocation information in the *TMS320C5x C Source Debugger User's Guide*.)
- Miscellaneous materials** Blank, formatted disks

Software checklist

- Operating system** MS-DOS or PC-DOS (version 3.0 or later)
Optional: Windows (version 3.0 or later)
- Software tools** TMS320C1x/C2x/C2xx/C5x assembler and linker
Optional: TMS320C2x/C2xx/C5x C compiler
- Optional files included with the debugger package**

siminit.cmd is a general-purpose batch file that contains debugger commands. This batch file, shipped with the debugger, defines a 'C5x memory map. When you first start using the debugger, this memory map should be sufficient for your needs. Later, you may want to define your own memory map. For information about defining your own memory map, refer to the defining a memory map chapter in the *TMS320C5x C Source Debugger User's Guide*. If a memory map batch file isn't present when you invoke the debugger, all memory is invalid initially.

init.clr is a general-purpose screen configuration file. If *init.clr* isn't present when you invoke the debugger, the debugger uses the default screen configuration.

init.25, *init.43*, and *init.50* have been provided for basic 80 x 25, 80 x 43, and 80 x 50 screen sizes, respectively. The *init.clr* file brings up the debugger in 80 x 25 mode. To bring the debugger up in another mode, copy one of the *init.xx* files to the *init.clr* file. When you first invoke the debugger, the default screen configuration should be sufficient for your needs. Later, you may want to define your own custom configuration.

The default configuration is for color monitors; an additional file, *mono.clr*, can be used for monochrome monitors. When you first start to use the debugger, the default screen configuration should be sufficient for your needs. Later, you may want to define your own custom configuration.

For information about these files and about setting up your own screen configuration, refer to customizing the debugger display section in the *TMS320C5x C Source Debugger User's Guide*.
-
-
-

1.2 Step 1: Installing the Simulator and Debugger Software

This section explains how to install the simulator and debugger on a hard-disk system.

- 1) Make a backup copy of the product disk(s). The product disk(s) include both a DOS and a Windows version of the debugger and simulator.
- 2) On your hard disk, create a directory named sim5x. This directory will contain the 'C5x software.

```
MD C:\sim5x
```

- 3) Insert the product disk(s) into drive A. Copy the contents of the disk(s).

```
COPY A:\*.* C:\sim5x\*.* /V
```

The DOS version of the debugger is named sim5x.exe. The Windows version of the debugger is named sim5xw.exe. Throughout this document, the debugger is referred to simply as sim5x.

1.3 Step 2: Setting Up the Debugger Environment

To ensure that your debugger works correctly, you must identify the items that are listed in Table 1–1. You can specify this information either in an initialization batch file (see page 1-6) or in your *autoexec.bat* file (see page 1-7).

Table 1–1. Debugger Environment Variables

To identify . . .	Use a statement with this format . . .
Directory with executable files for the C source debugger	PATH=drive :\ directory
Directory with debugger data files, such as <i>init.cmd</i> and <i>init.clr</i>	SET D_DIR=drive :\ directory
Directory with the program source files that you want to debug	SET D_SRC=drive :\ directory
Address of the emulator port on your PC and other options that you want to use every time that you invoke the debugger	SET D_OPTIONS= [object filename] [options]
	<p>[object filename] Names the file that you want to load every time that you invoke the debugger.</p> <p>[options] Indicates the port address and other options; for more information, see the <i>TMS320C5x C Source Debugger User's Guide</i>.</p> <p>-p port address Identifies the emulator port on your PC:</p> <p>378 default port address for the XDS510PP (LPT1 on most PCs); to verify the address of the printer port where you connected the XDS510PP, see your PC documentation.</p> <p>240 default port address for the XDS510; for more information, see the <i>XDS51x Emulator Installation Guide</i>.</p> <p>-b Selects a screen size of 80 characters by 43 lines (EGA or VGA)</p> <p>-bb Selects a screen size of 80 characters by 50 lines (VGA only)</p> <p>-font height Uses the Windows Terminal font closest in point size to the specified height</p> <p>-i pathname Identifies additional directories</p> <p>-mv version Specifies the memory map to use with the simulator</p> <p>-profile Allows you to enter the profiling environment</p> <p>-s Loads only the symbol table for a named object file</p> <p>-t filename Identifies a new initialization file</p> <p>-v Loads the object code with a minimal symbol table</p>

Note: When you invoke the debugger, you can include `-x` on the command line to override any `D_OPTIONS` in the initialization file or in your *autoexec.bat* file.

Defining an initialization batch file

To create an initialization file named *initdb.bat*, follow these steps using a text editor; be careful that no spaces precede the equal (=) sign wherever it appears.

- 1) To specify the location of the C source debugger executable files and to ensure that this statement does not overwrite PATH statements in other batch files, type:

```
PATH=C:\sim5x;%PATH%
```

- 2) To specify the directory with the C source debugger data files, type:

```
SET D_DIR=C:\sim5x
```

- 3) To specify directories that contain the program source files that you want to debug, use the following format to set the D_SRC environment variable:

```
SET D_SRC=pathname1[;pathname2...]
```

- 4) To specify the emulator port address and other options, use the following format to define the D_OPTIONS environment variable:

```
SET D_OPTIONS=[-p port_address] [ options ]
```

- 5) To add the emulator-reset command to the file, type:

```
EMURST
```

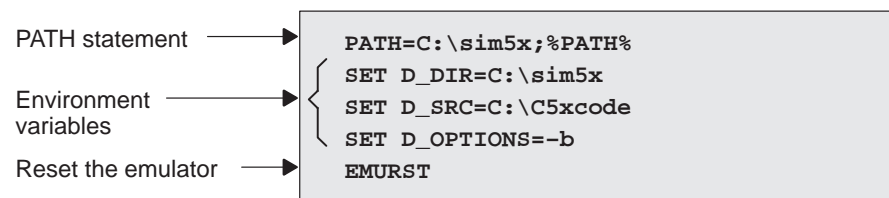
- 6) Save the file as *initdb.bat*, and then exit the text editor.

- 7) Before you start Windows and any time that you power up or reboot your PC, invoke this file from a DOS prompt by entering:

```
INITDB 
```

Figure 1–1 shows a sample initialization file that contains the required path, environment variables, and emulator-reset statement.

Figure 1–1. Sample Initialization Batch File



Changing the autoexec.bat file

If you are sure that no programs will be affected by changing your autoexec.bat file, you can specify the debugger environment in that file. To change your autoexec.bat file, follow these steps using a text editor; be careful that no spaces precede the equal (=) sign wherever it appears.


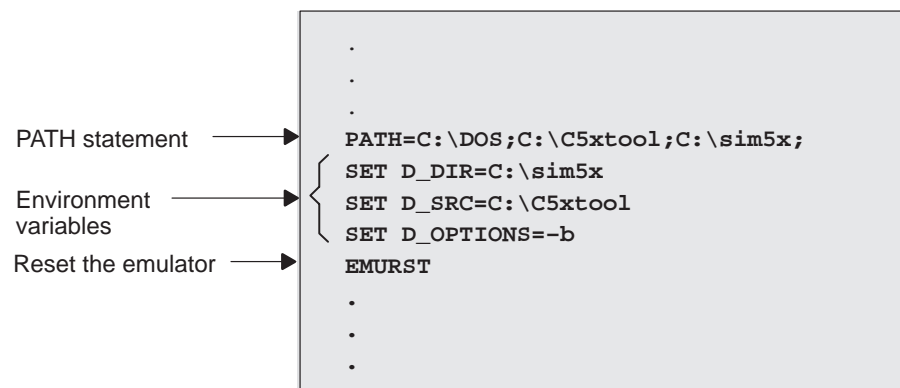
- 1) At the end of the PATH statement in your autoexec.bat file, type:
`;C:\sim5x;`
- 2) To specify the directory that contains the C source debugger files, type:
`SET D_DIR=C:\sim5x`
- 3) To specify directories that contain the program source files that you want to debug, use the following format to set the D_SRC environment variable:
`SET D_SRC=pathname1 [;pathname2 . . .]`
- 4) To specify the emulator port address and other options, use the following format to define the D_OPTIONS environment variable:
`SET D_OPTIONS=[-p port_address] [options]`
- 5) To add the emulator-reset command to the file, type:
`EMURST`
- 6) Save the file, and then exit the text editor.
- 7) Before you invoke the debugger for the first time, invoke the autoexec.bat file from an DOS prompt by entering:
`AUTOEXEC` 

Figure 1–2 shows a portion of the autoexec.bat file with the required path, environment variables, and emulator-reset statement.

Figure 1–2. Sample autoexec.bat File



1.4 Step 3: Verifying the Installation

To ensure that you have correctly installed the simulator and debugger software, enter this command at the system prompt:

```
sim5x c:\sim5x\sample
```

You should see a display similar to this one:

The screenshot shows the TMS320C5x simulator debugger interface. It features a menu bar at the top with options: Load, Break, Watch, Memory, Color, MoDe, Analysis, Pin, Run=F5, Step=F8, Next=F10. The main display is divided into four panes:

- DISASSEMBLY:** Shows assembly instructions for the function `c_int0:`.

Address	Hex	Op	Op2	Op3	Op4
20cf	bf08	LAR	AR0	#08a1h	
20d1	bf09	LAR	AR1	#00a1h	
20d3	bf00	SPM		0	
20d4	be47	SETC	SXM		
20d5	bf80	LACC	#2143h		
20d7	b801	ADD	#1		
20d8	e388	BCND	20dch,EQ		
20da	7a89	CALL	20e0h,* ,AR1		
20dc	7a89	CALL	main,* ,AR1		
20de	7a89	CALL	abort,* ,AR1		
20e0	bf80	LACC	#2143h		
20e2	8bc00	LDP	#0		
20e3	a680	TBLR	*		
20e4	b801	ADD	#1		
20e5	028a	LAR	AR2,* ,AR2		
- CPU:** Shows the current state of CPU registers.

Register	Value	Register	Value
ACC	0000005f	TOS	005d
ACCB	01ff01ff	AR0	08ab
PREG	00000005	AR1	08ac
PC	20cf	AR2	08a5
AR0	08ab	AR3	00a3
AR2	08a5	AR4	00a4
AR3	00a3	AR5	0807
AR4	00a4	AR6	08a4
AR5	0807	AR7	00a7
AR6	08a4	ST0	2610
AR7	00a7	ST1	cdfc
ST0	2610	PMST	0038
ST1	cdfc	TIM	249d
PMST	0038	IMR	01ff
TIM	249d	IFR	0008
IMR	01ff	DBMR	0000
IFR	0008	BMAR	5555
DBMR	0000	INDX	08ab
BMAR	5555	TRG0	ffe1
INDX	08ab	TRG1	ffe1
TRG0	ffe1	TRG2	fff1
TRG1	ffe1	SPCR	0800
TRG2	fff1	TCR	0000
SPCR	0800		
TCR	0000		
- COMMAND:** Shows the execution of the `sim5x` command.

```
TMS320C5x Revision 1
Loading sample.out
 34 Symbols loaded
Done
>>>
```
- MEMORY:** Shows a memory dump.

Address	Hex	Hex	Hex	Hex	Hex	Hex	Hex	Hex	Hex
0000	0000	0000	0000	0000	01ff	ff00	0008	0038	
0008	0000	0000	20f1	20f3	0001	ffe1	fff1	0000	
0010	08ab	08ac	08a5	00a3	0004	0807	08a4	00a7	
0018	08ab	08ab	0000	0000	0000	0000	ffe7	5555	
0020	0000	0000	0000	0000	249d	ffff	0000	0000	
0028	ffff	ffff	000f	0000	0000	0000	0000	0000	

- If you see a display similar to this one, you have correctly installed your simulator and debugger.
- If you don't see a similar display, your debugger or simulator may not be installed properly. Go back through the installation instructions and be sure that you have followed each step correctly; then reenter the command above.

1.5 Using the Debugger With Windows

If you're using Windows, you can freely move or resize the debugger display on the screen. If the resized display is bigger than the debugger requires, the extra space is not used. If the resized display is smaller than required, the display is clipped. Note that when the display is clipped, it can't be scrolled.

You may want to create an icon to make it easier to invoke the debugger from within the Windows environment. Refer to your Windows manual for details.

You should run Windows in either the standard mode or the 386-enhanced mode to get the best results.

Installing the Simulator and C Source Debugger With SunOS

This chapter describes how to install the 'C5x simulator and the C source debugger on a SPARCstation running SunOS™ or Solaris™. When you complete the installation, turn to the *TMS320C5x C Source Debugger User's Guide*.

Topic	Page
2.1 What You Need	2-2
2.2 Step 1: Installing the Simulator and Debugger Software	2-4
2.3 Step 2: Setting Up the Debugger Environment	2-6
2.4 Step 3: Verifying the Installation	2-8
2.5 Using the Debugger With the X Window System	2-9

2.1 What You Need

To install the 'C5x C source debugger and simulator, you need the items in the following hardware and software checklists.

Hardware checklist

- | | | |
|--------------------------|--------------------------|--|
| <input type="checkbox"/> | Host | A SPARCstation or a system that is 100% compatible with a SPARCstation 2 class or higher |
| <input type="checkbox"/> | Display | Monochrome or color monitor (color recommended) |
| <input type="checkbox"/> | Disk space | 2M bytes of disk space |
| <input type="checkbox"/> | Required hardware | CD-ROM drive |
| <input type="checkbox"/> | Optional hardware | Mouse |

Software checklist

- Operating system** OpenWindows™ version 3.0 (or higher) running under SunOS version 4.1.3 (or higher) or SunOS version 5.x (also known as Solaris 2.x).
- Root privileges** If you are running SunOS 4.1.x, 5.0, or 5.1, you *must* have root privileges to mount and unmount the CD-ROM. If you don't, get help from your system administrator.
- Software tools** TMS320C1x/C2x/C2xx/C5x assembler and linker
Optional: TMS320C2x/C2xx/C5x C compiler
- Optional files included with the debugger package** *siminit.cmd* is a general-purpose batch file that contains debugger commands. This batch file, shipped with the debugger, defines a 'C5x memory map. When you first start using the debugger, this memory map should be sufficient for your needs. Later, you may want to define your own memory map. For information about defining your own memory map, refer to the defining a memory map chapter in the *TMS320C5x C Source Debugger User's Guide*. If this memory map batch file isn't present when you invoke the debugger, all memory is invalid initially.
- init.clr* is a general-purpose screen configuration file. If *init.clr* isn't present when you invoke the debugger, the debugger uses the default screen configuration.
- init.25*, *init.43*, and *init.50* have been provided for basic 80 x 25, 80 x 43, and 80 x 50 screen sizes, respectively. The *init.clr* file brings up the debugger in 80 x 25 mode. To bring the debugger up in another mode, copy one of the *init.xx* files to the *init.clr* file.
- The default configuration is for color monitors; an additional file, *mono.clr*, can be used for monochrome monitors. When you first start to use the debugger, the default screen configuration should be sufficient for your needs. Later, you may want to define your own custom configuration.

For information about these files and about setting up your own screen configuration, refer to the customization information in the *TMS320C5x C Source Debugger User's Guide*.

2.2 Step 1: Installing the Simulator and Debugger Software

This section explains how to install the simulator and debugger software on your hard-disk system. The software package is shipped on a CD-ROM. To install the software, you must mount the CD-ROM, copy the files, and unmount the CD-ROM.

Mounting the CD-ROM

Note: Root Privileges




If you are running SunOS 4.1.x, 5.0, or 5.1, you *must* have root privileges to mount the CD-ROM. If you don't, get help from your system administrator.

The steps to mount the CD-ROM vary according to your operating-system version. Follow the appropriate steps below:

- If you have SunOS 4.1.x, load the CD-ROM into the drive and enter the following from a command shell:


```
mount -rt hsfs /dev/sr0 /cdrom   
exit   
cd /cdrom/sparc 
```

- If you have SunOS 5.0 or 5.1, load the CD-ROM into the drive and enter the following from a command shell:


```
mount -rF hsfs /dev/sr0 /cdrom   
exit   
cd /cdrom/cdrom0/sparc 
```

- If you have SunOS 5.2 or higher:

- If your CD-ROM drive is already attached, load the CD-ROM into the drive and enter the following from a command shell:

```
cd /cdrom/cdrom0/sparc 
```

- If you do not have a CD-ROM drive attached, you must shut down your system to the PROM level, attach the CD-ROM drive, and enter the following:

```
boot -r 
```

After you log into your system, load the CD-ROM into the drive and enter the following from a command shell:

```
cd /cdrom/cdrom0/sparc 
```

Copying the files

After you mount the CD-ROM, you must create the directory that will contain the debugger software and copy the software to that directory.

- 1) Create a directory named *sim5x* on your hard disk. To create this directory, enter:

```
mkdir /your_pathname/sim5x
```

- 2) Copy the files from the CD-ROM to your hard-disk system:

```
cp -r * /your_pathname/sim5x
```

Unmounting the CD-ROM

Note: Root Privileges

If you are running SunOS 4.1.x, 5.0, or 5.1, you *must* have root privileges to unmount the CD-ROM. If you don't, get help from your system administrator.

You must unmount the CD-ROM after copying the files.

- If you have SunOS 4.1.x, 5.0, or 5.1, enter the following from a command shell:

```
cd  
umount /cdrom  
eject /dev/sr0  
exit
```

- If you have SunOS 5.2 or higher, enter the following from a command shell:

```
cd  
eject
```

2.3 Step 2: Setting Up the Debugger Environment

To ensure that your debugger works correctly, you must identify the items that are listed in Table 2–1. You specify this information in your shell configuration file in your home directory (for example, the `.cshrc` file for a C shell). After modifying your shell configuration file, you must reinitialize it.

Table 2–1. Debugger Environment Variables

To identify . . .	Use a statement with this format . . .
Directory with executable files for the C source debugger	set path = (. /directory)
Directory with debugger data files, such as <i>init.cmd</i> and <i>init.clr</i>	setenv D_DIR "/directory"
Directory with the program source files that you want to debug	setenv D_SRC "/directory"
For an X Window system, display the debugger on a different machine (see Section 2.5, <i>Using the Debugger With the X Window System</i> , on page 2-9)	setenv DISPLAY "machinename"
Address of the emulator port on your PC and other options that you want to use every time that you invoke the debugger	setenv D_OPTIONS [object filename] [options] [object filename] Names the file that you want to load every time that you invoke the debugger. [options] Indicates the port address and other options; for more information, see the <i>TMS320C5x C Source Debugger User's Guide</i> . -b Selects a screen size of 80 characters by 43 lines (EGA or VGA) -bb Selects a screen size of 80 characters by 50 lines (VGA only) -d machine For an X Window system, display the debugger on a different machine. Use instead of the DISPLAY environment variable. -i pathname Identifies additional directories -mv version Specifies the memory map to use with the simulator -profile Allows you to enter the profiling environment -s Loads only the symbol table for a named object file -t filename Identifies a new initialization file -v Loads the object code with a minimal symbol table

Note: When you invoke the debugger, you can include `-x` on the command line to override any `D_OPTIONS` in the initialization file or in your `autoexec.bat` file.

Figure 2–1. Sample Shell Configuration File for an X Window System

```
set path statement → set path = (. /bin /usr/ucb /usr/contrib/bin /usr/bin \  
                    /usr/openwin/bin /user/fred/sim5x)  
Environment variables → { setenv D_DIR "/user/fred/sim5x"  
                          setenv D_SRC "/user/fred/C5xsource"  
                          setenv DISPLAY "barney:0"  
                          setenv D_OPTIONS "-b"  
Reset the emulator → emurst
```

Reinitializing your shell

When you modify your shell configuration file, you must ensure that the changes are made to your current session. For example, if you are using a C shell, use this command to reread the `.cshrc` file:

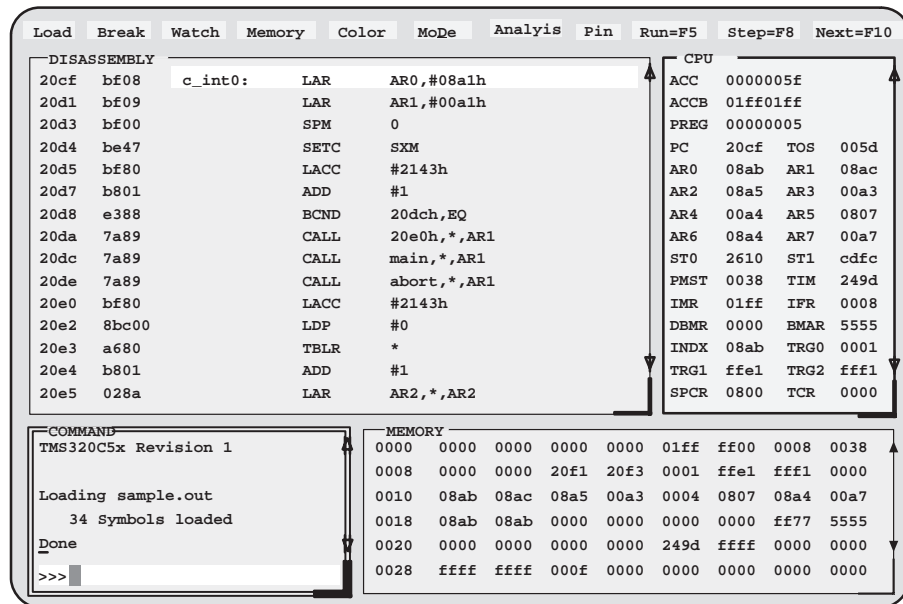
```
source ~/.cshrc 
```

2.4 Step 3: Verifying the Installation

To ensure that you have correctly installed the simulator and debugger software, enter this command at the system prompt:

```
sim5x sample
```

You should see a display similar to this one:



- If you see a display similar to this one, you have correctly installed your simulator and debugger.
- If you don't see a similar display, then your debugger or simulator may not be installed properly. Go back through the installation instructions and be sure that you have followed each step correctly; then reenter the command above.

2.5 Using the Debugger With the X Window System

If you're using the X Window System to run the 'C5x debugger, you need to know about the keyboard's special keys, the debugger fonts, and using the debugger on a monochrome monitor.

Using the keyboard's special keys

The debugger uses some special keys that you can map differently from your particular keyboard. Some keyboards, such as the Sun Type 5 keyboard, may have these special symbols on separate keys. Other keyboards, such as the Sun Type 4 keyboard, do not have the special keys.

The special keys that the debugger uses are shown in the following table with their corresponding keysym. A *keysym* is a label that interprets a keystroke; it allows you to modify the action of a key on the keyboard.

Key	Keysym
(F1) to (F10)	F1 to F10
(PAGE UP)	Prior
(PAGE DOWN)	Next
(HOME)	Home
(END)	End
(INSERT)	Insert
(→)	Right
(←)	Left
(↑)	Up
(↓)	Down

Use the X utility `xev` to check the keysyms that are associated with your keyboard. If you need to change the keysym definitions, use the `xmodmap` utility. For example, you could create a file that contains the following commands and use that file with `xmodmap` to change a Sun Type 4 keyboard to match the keys listed above:

```
keysym R13      = End
keysym Down    = Down
keysym F35     = Next
keysym Left    = Left
keysym Right   = Right
keysym F27     = Home
keysym Up      = Up
keysym F29    = Prior
keysym Insert  = Insert
```

Refer to your X Window System documentation for more information about using `xev` and `xmodmap`.

Changing the debugger font

You can change the font of the debugger screen by using the `xrdb` utility and modifying the `.Xdefaults` file in your root directory. For example, to change the 'C5x debugger fonts to Courier, add the following line to the `.Xdefaults` file:

```
sim5x*font:courier
```

For more information about using `xrdb` to change the font, refer to your X Window System documentation.

Color mappings on monochrome screens

Although a color monitor is recommended, the following table shows the color mappings for monochrome screens:

Color	Appearance on Monochrome Screen
black	black
blue	black
green	white
cyan	white
red	black
magenta	black
yellow	white
white	white

Installing the Simulator and C Source Debugger With HP-UX

This chapter describes how to install the 'C5x simulator and the C source debugger on a HP9000 series 700™ PA-RISC™ system running HP-UX™. When you complete the installation, turn to the *TMS320C5x C Source Debugger User's Guide*.

Topic	Page
3.1 What You Need	3-2
3.2 Step 1: Installing the Simulator and Debugger Software	3-4
3.3 Step 2: Setting Up the Debugger Environment	3-6
3.4 Step 3: Verifying the Installation	3-8
3.5 Using the Debugger With the X Window System	3-9

3.1 What You Need

To install the 'C5x C source debugger and simulator you need the items in the following hardware and software checklists.

Hardware checklist

- | | | |
|--------------------------|--------------------------|---|
| <input type="checkbox"/> | Host | An HP9000 Series 700 PA-RISC system |
| <input type="checkbox"/> | Display | Monochrome or color (color recommended) |
| <input type="checkbox"/> | Disk space | 2M bytes of disk space |
| <input type="checkbox"/> | Required hardware | CD-ROM drive |
| <input type="checkbox"/> | Optional hardware | Mouse |

Software checklist

- Operating system** HP-UX 9.x or higher.
- Root privileges** You *must* have root privileges to mount and unmount the CD-ROM. If you don't, get help from your system administrator.
- Software tools** TMS320C1x/C2x/C2xx/C5x assembler and linker
Optional: TMS320C2x/C2xx/C5x C compiler
- Optional files included with the debugger package**

siminit.cmd is a general-purpose batch file that contains debugger commands. This batch file, shipped with the debugger, defines a 'C5x memory map. When you first start using the debugger, this memory map should be sufficient for your needs. Later, you may want to define your own memory map. For information about defining your own memory map, refer to the defining a memory map chapter in the *TMS320C5x C Source Debugger User's Guide*. If a memory map batch file isn't present when you invoke the debugger, all memory is invalid initially.

init.clr is a general-purpose screen configuration file. If *init.clr* isn't present when you invoke the debugger, the debugger uses the default screen configuration.

init.25, *init.43*, and *init.50* have been provided for basic 80×25, 80×43, and 80×50 screen sizes, respectively. The *init.clr* file brings up the debugger in 80×25 mode. To bring the debugger up in another mode, copy one of the *init.xx* files to the *init.clr* file.

The default configuration is for color monitors; an additional file, *mono.clr*, can be used for monochrome monitors. When you first start to use the debugger, the default screen configuration should be sufficient for your needs. Later, you may want to define your own custom configuration.

For information about these files and about setting up your own screen configuration, refer to the customization information in the *TMS320C5x C Source Debugger User's Guide*.
-
-
-

3.2 Step 1: Installing the Simulator and Debugger Software

This section explains how to install the simulator and debugger software on your hard-disk system. The software package is shipped on a CD-ROM. To install the software, you must mount the CD-ROM, copy the files, and unmount the CD-ROM.

Mounting the CD-ROM



Note: Root Privileges

You *must* have root privileges to mount the CD-ROM. If you don't, get help from your system administrator.

You can mount the CD-ROM using the UNIX mount command or the SAM (system administration manager):

- To use the UNIX mount command, follow these steps:

- 1) To mount the CD-ROM, enter:

```
mount -rt cdfs /dev/dsk/your_cdrom_device /cdrom   
exit 
```

- 2) Make the hp700 directory on the CD-ROM the current directory. For example, if the CD-ROM is mounted at /cdrom, enter:

```
cd /cdrom/hp700 
```

- To use SAM to mount the CD-ROM, see *System Administration Tasks*, the HP documentation about SAM, for instructions.

Copying the files

After you mount the CD-ROM, you must create the directory that will contain the debugger software and copy the software to that directory.

- 1) Create a directory named *sim5x* on your hard disk. To create this directory, enter:

```
mkdir /your_pathname/sim5x 
```

- 2) Copy the files from the CD-ROM to your hard-disk system:

```
cp -r * /your_pathname/sim5x 
```

Unmounting the CD-ROM

Note: Root Privileges

You *must* have root privileges to unmount the CD-ROM. If you don't, get help from your system administrator.

You must unmount the CD-ROM after copying the files. Enter:

```
cd [2]  
umount /cdrom [2]  
exit [2]
```

3.3 Step 2: Setting Up the Debugger Environment

To ensure that your debugger works correctly, you must identify the items that are listed in Table 3–1. You specify this information in your shell configuration file in your home directory (for example, the `.cshrc` file for a C shell). After modifying your shell configuration file, you must reinitialize it.

Table 3–1. Debugger Environment Variables

To identify . . .	Use a statement with this format . . .
Directory with executable files for the C source debugger	set path = (. /directory)
Directory with debugger data files, such as <i>init.cmd</i> and <i>init.clr</i>	setenv D_DIR "/directory"
Directory with the program source files that you want to debug	setenv D_SRC "/directory"
For an X Window system, display the debugger on a different machine (see Section 3.5, <i>Using the Debugger With the X Window System</i> , on page 3-9)	setenv DISPLAY "machinename"
Address of the emulator port on your PC and other options that you want to use every time that you invoke the debugger	setenv D_OPTIONS [object filename] [options] [object filename] Names the file that you want to load every time that you invoke the debugger. [options] Indicates the port address and other options; for more information, see the <i>TMS320C5x C Source Debugger User's Guide</i> . -b Selects a screen size of 80 characters by 43 lines (EGA or VGA) -bb Selects a screen size of 80 characters by 50 lines (VGA only) -d machine For an X Window system, display the debugger on a different machine. Use instead of the DISPLAY environment variable. -i pathname Identifies additional directories -mv version Specifies the memory map to use with the simulator -profile Allows you to enter the profiling environment -s Loads only the symbol table for a named object file -t filename Identifies a new initialization file -v Loads the object code with a minimal symbol table

Note: When you invoke the debugger, you can include `-x` on the command line to override any `D_OPTIONS` in the initialization file or in your `autoexec.bat` file.

Figure 3–1. Sample Shell Configuration File for an X Window System

```
set path statement → set path = (. /bin /usr/ucb /usr/contrib/bin /usr/bin \  
                    /usr/openwin/bin /user/fred/sim5x)  
Environment variables → { setenv D_DIR "/user/fred/sim5x"  
                        setenv D_SRC "/user/fred/C5xsource"  
                        setenv DISPLAY "barney:0"  
                        setenv D_OPTIONS "-b"  
Reset the emulator → emurst
```

Reinitializing your shell

When you modify your shell configuration file, you must ensure that the changes are made to your current session. For example, if you are using a C shell, use this command to reread the `.cshrc` file:

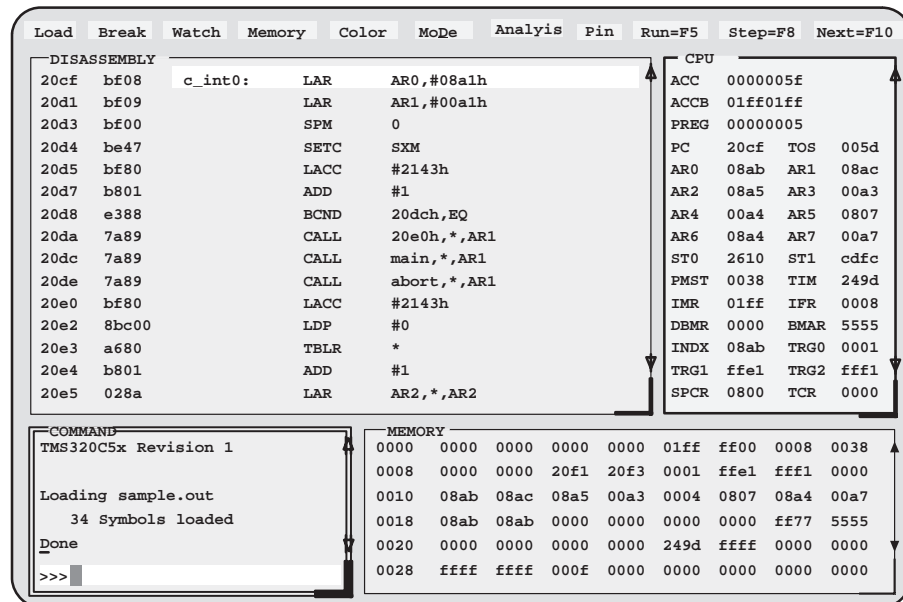
```
source ~/.cshrc 
```

3.4 Step 3: Verifying the Installation

To ensure that you have correctly installed the simulator and debugger software, enter this command at the system prompt:

```
sim5x sample
```

You should see a display similar to this one:



- If you see a display similar to this one, you have correctly installed your simulator and debugger.
- If you don't see a similar display, then your debugger or simulator may not be installed properly. Go back through the installation instructions and be sure that you have followed each step correctly; then reenter the command above.

3.5 Using the Debugger With the X Window System

If you're using the X Window System to run the 'C5x debugger, you need to know about the keyboard's special keys, the debugger fonts, and using the debugger on a monochrome monitor.

Using the keyboard's special keys

The debugger uses some special keys that you can map differently from your particular keyboard. Some keyboards may have these special symbols on separate keys. Other keyboards do not have the special keys.

The special keys that the debugger uses are shown in the following table with their corresponding keysym. A *keysym* is a label that interprets a keystroke; it allows you to modify the action of a key on the keyboard.

Key	Keysym
(F1) to (F10)	F1 to F10
(PAGE UP)	Prior
(PAGE DOWN)	Next
(HOME)	Home
(END)	End
(INSERT)	Insert
→	Right
←	Left
↑	Up
↓	Down

Use the X utility `xev` to check the keysyms that are associated with your keyboard. If you need to change the keysym definitions, use the `xmodmap` utility. For example, you could create a file that contains the following commands and use that file with `xmodmap` to change a keyboard to match the keys listed above:

```
keysym R13 = End
keysym Down = Down
keysym F35 = Next
keysym Left = Left
keysym Right = Right
keysym F27 = Home
keysym Up = Up
keysym F29 = Prior
keysym Insert = Insert
```

Refer to your X Window System documentation for more information about using `xev` and `xmodmap`.

Changing the debugger font

You can change the font of the debugger screen by using the `xrdb` utility and modifying the `.Xdefaults` file in your root directory. For example, to change the fonts of the 'C5x debugger to Courier, add the following line to the `.Xdefaults` file:

```
sim5x*font:courier
```

For more information about using `xrdb` to change the font, refer to your X Window System documentation.

Color mappings on monochrome screens

Although a color monitor is recommended, the following table shows the color mappings for monochrome screens:

Color	Appearance on Monochrome Screen
black	black
blue	black
green	white
cyan	white
red	black
magenta	black
yellow	white
white	white

Release Notes

This chapter contains information that has changed for the simulator version of the TMS320C5x C source debugger. This information became available after the release of the *TMS320C5x C Source Debugger User's Guide*.

Topic	Page
4.1 Changes to the Options Used to Invoke the C Source Debugger ..	4-2
4.2 Changes to Defining a Memory Map	4-2
4.3 Known Problem	4-2

4.1 Changes to the Options Used to Invoke the C Source Debugger

In the *TMS320C5x C Source Debugger User's Guide*, Section 1.8, *Debugger Options*, describes the options that you can use to invoke the C source debugger. The `-mv` option has been modified for the simulator version of the debugger.

The `-mv` option tells the simulator which memory map to use as described in the following table:

Option	Device Simulated	Peripherals Simulated
<code>-mv50</code>	'C50	Serial port and TDM serial port
<code>-mv51</code>	'C51	Serial port and TDM serial port
<code>-mv53</code>	'C53	Serial port and TDM serial port
<code>-mv56</code>	'C56	Serial port and buffered serial port
<code>-mv57</code>	'C57	Serial port and buffered serial port

4.2 Changes to Defining a Memory Map

In the *TMS320C5x C Source Debugger User's Guide*, Chapter 6, *Defining a Memory Map*, explains how to define a memory map. The current version of the simulator has enhanced features for simulating peripherals and serial ports. Due to these enhancements to the simulator version of the C source debugger, Chapter 6 required many modifications and additions.

Chapter 5, *Defining a Memory Map*, in this manual entirely replaces Chapter 6 in the *TMS320C5x C Source Debugger User's Guide*.

4.3 Known Problem

For all systems, the following instructions move the BPT pseudoinstruction (opcode 0xeb3) if a breakpoint is set at the program memory address operand of these instructions:

- TBLR—table read
- BLPD—block move from program memory to data memory

Defining a Memory Map

Before you begin a debugging session, you must supply the debugger with a memory map. The memory map tells the debugger which areas of memory it can and can't access. Note that the commands described in this chapter can also be entered by using the Memory pulldown menu.

Topic	Page
5.1 The Memory Map: What It Is and Why You Must Define It	5-2
5.2 A Sample Memory Map	5-4
5.3 Identifying Usable Memory Ranges	5-6
5.4 Enabling Memory Mapping	5-8
5.5 Checking the Memory Map	5-9
5.6 Modifying the Memory Map During a Debugging Session	5-10
5.7 Using Multiple Memory Maps for Multiple Target Systems	5-11
5.8 Simulating I/O Space (Simulator Only)	5-12
5.9 Simulating External Interrupts (Simulator Only)	5-14
5.10 Simulating Peripherals (Simulator Only)	5-18
5.11 Simulating Standard Serial Ports (Simulator Only)	5-19
5.12 Simulating Buffered Serial Ports (Simulator Only)	5-22
5.13 Simulating TDM Serial Ports (Simulator Only)	5-25

5.1 The Memory Map: What It Is and Why You Must Define It

A memory map tells the debugger which areas of memory it can and can't access. Memory maps vary, depending on the application. Typically, the map matches the MEMORY definition in your linker command file.

Note:

When the debugger compares memory accesses against the memory map, it performs this checking in software, not hardware. The debugger can't prevent your program from attempting to access nonexistent memory.

A special default initialization batch file included with the debugger package defines a memory map for your version of the debugger. This memory map may be sufficient when you first begin using the debugger. However, the debugger provides a complete set of memory-mapping commands that let you modify the default memory map or define a new memory map.

You can define the memory map interactively by entering the memory-mapping commands while you're using the debugger. This can be inconvenient because, in most cases, you set up one memory map before you begin debugging and use this map for all of your debugging sessions. The easiest method for defining a memory map is to put the memory-mapping commands in a batch file.

Defining the memory map in a batch file

There are two methods for defining the memory map in a batch file:

- Redefine the memory map defined in the initialization batch file.
- Define a memory map in a separate batch file of your own.

When you invoke the debugger, it follows these steps to find the batch file that defines your memory map:

- 1) The debugger checks if you've used the `-t` debugger option. If the debugger finds the `-t` option, it executes the specified file. (Use the `-t` option to specify a batch file other than the initialization batch file shipped with the debugger.)
- 2) If you don't use the `-t` option, the debugger looks for the default initialization batch file called *init.cmd*. The debugger reads and executes the commands in the file.

Potential memory map problems

You may experience these problems if the memory map isn't correctly defined and enabled:

- Accessing invalid memory addresses.** If you don't supply a batch file containing memory-map commands, then the debugger is initially unable to access any target memory locations. Invalid memory addresses and their contents are highlighted in the data-display windows. (On color monitors, invalid memory locations, by default, are displayed in red.)
- Accessing an undefined or protected area.** When memory mapping is enabled, the debugger checks each of its memory accesses against the memory map. If you attempt to access an undefined or protected area, the debugger displays an error message. For specific error messages, see Appendix D, *Debugger and PDM Messages*.
- Loading a COFF file with sections that cross a memory range.** Be sure that the map ranges you specify in a COFF file match those that you define with the MA command (described on page 5-6). Alternatively, you can turn memory mapping off during a load by using the MAP OFF command (see page 5-8).

5.2 A Sample Memory Map

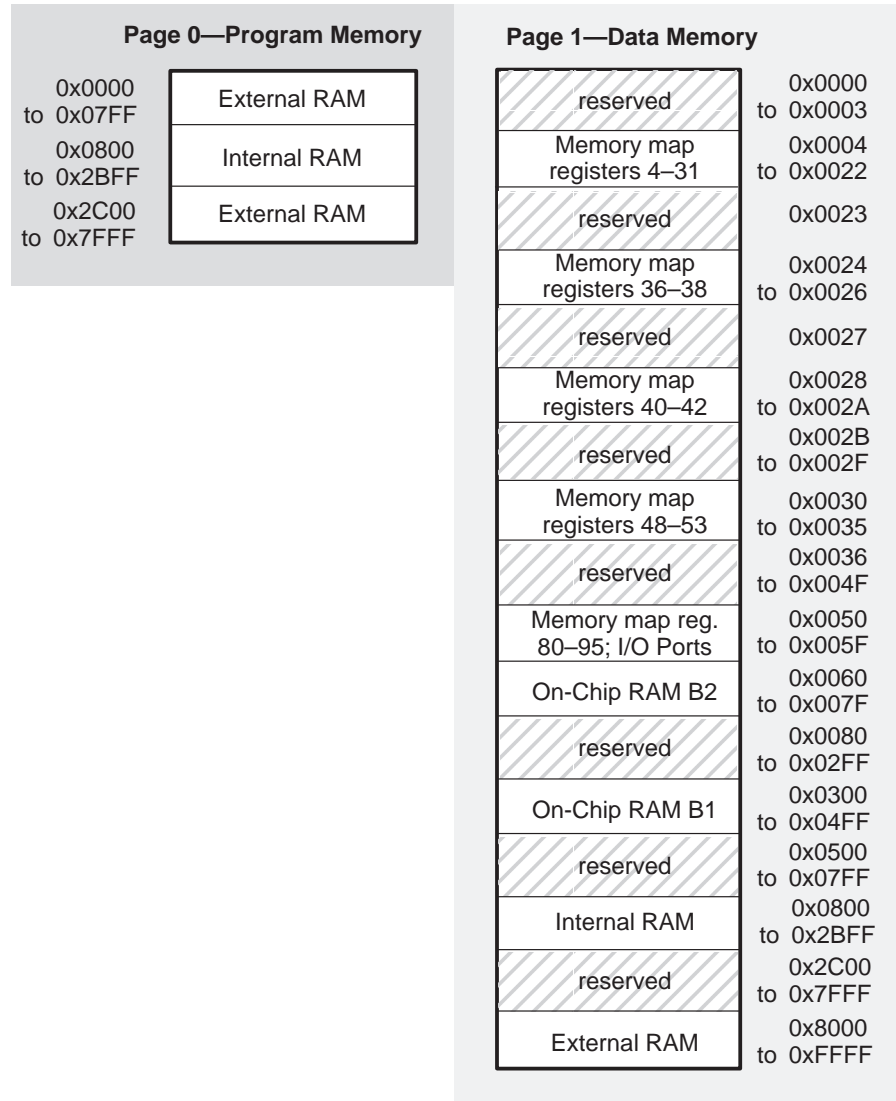
Because you must define a memory map before you can run any programs, it's convenient to define the memory map in the initialization batch files. Figure 5–1 shows the memory map commands that are defined in the initialization batch file that accompanies the EVM. If you are using the EVM, you can use the file as is, edit it, or create your own memory map batch file. The files shipped with the simulator and emulator are similar to that of the EVM.

Figure 5–1. Memory Map Commands in the Sample Initialization Batch File for the EVM

```
MA 0x0000, 0, 0x0800, RAM
MA 0x0800, 0, 0x2400, RAM
MA 0x2c00, 0, 0x5400, RAM
MA 0x0004, 1, 0x001f, RAM
MA 0x0024, 1, 0x0003, RAM
MA 0x0028, 1, 0x0003, RAM
MA 0x0030, 1, 0x0006, RAM
MA 0x0050, 1, 0x0010, RAM
MA 0x0060, 1, 0x0020, RAM
MA 0x0300, 1, 0x0200, RAM
MA 0x0800, 1, 0x2400, RAM
MA 0x8000, 1, 0x8000, RAM
```

The MA (map add) commands define valid memory ranges and identify the read/write characteristics of the memory ranges. Figure 5–2 illustrates the memory map defined by the default initialization batch file.

Figure 5–2. Sample Memory Map for Use With a 'C5x EVM



5.3 Identifying Usable Memory Ranges



ma The debugger's MA (memory add) command identifies valid ranges of target memory. The syntax for this command is:

ma *address, page, length, type*

- address*—defines the starting address of a range. This parameter is an absolute address, any C expression, the name of a C function, or an assembly language label.

A new memory map must not overlap an existing entry. If you define a range that overlaps an existing range, the debugger ignores the new range and displays this error message in the COMMAND window display area:

```
Conflicting map range
```

- page*—a one-digit number that identifies the type of memory (program, data, or I/O) that the range occupies:

To identify this page,	Use this value as the <i>page</i> parameter
Program memory	0
Data memory	1
I/O space	2

- length*—defines the length of the range. This parameter can be any C expression.
- type*—identifies the read/write characteristics of the memory range. The *type* must be one of these keywords:

To identify this kind of memory,	Use this keyword as the <i>type</i> parameter
Read-only memory	R or ROM
Write-only memory	W or WOM
Read/write memory	R W or RAM
No-access memory	PROTECT
Input port	IPOINT
Output port	OPOINT
Input/output port	IOPORT

Notes:

- ❑ The debugger caches memory that is not defined as a port type (IPORT, OPORT, or IOPORT). For ranges that you don't want cached, be sure to map them as ports.
- ❑ When you are using the simulator, you can use the parameter values IPORT, OPORT, and IOPORT to simulate I/O ports. See Section 5.8, *Simulating I/O Space*, on page 5-12.
- ❑ Be sure that the map ranges that you specify in a COFF file match those that you define with the MA command. Moreover, a command sequence such as:

```
ma x,0,y,ram; ma x+y,0,z,ram
```

doesn't equal

```
ma x,0,y+z,ram
```

If you plan to load a COFF block that spans the length of $y + z$, you should use the second MA command example. Alternatively, you can turn memory mapping off during a load by using the MAP OFF command (see Section 5.4, *Enabling Memory Mapping*, on page 5-8).

Memory mapping with the simulator (PC version)

The 'C5x simulator has memory-cache capabilities that allow you to allocate as much memory as you need. However, to use memory cache capabilities effectively with the 'C5x, do not allocate more than 20K words of memory in your memory map. For example, the memory map shown in Example 5–1 allocates 64K words of 'C5x program memory.

Example 5–1. Sample Memory Map for the TMS320C5x Using Memory-Cache Capabilities

```
MA 0,0,0x5000,R|W
MA 0x5000,0,0x5000,R|W
MA 0xa000,0,0x5000,R|W
MA 0xf000,0,0x1000,R|W
```

The simulator creates temporary files in a separate directory on your disk. For example, when you enter an MA (memory add) command, the simulator creates a temporary file in the root directory of your current disk. Therefore, if you are currently running your simulator on the C drive, temporary files are placed in the C:\ directory. This prevents the processor from running out of memory space while you are executing the simulator.

Note:

If you execute the simulator from a floppy drive (for example, drive A), the temporary files created in the A:\ directory.

All temporary files are deleted when you exit the simulator using the QUIT command. If, however, you exit the simulator with a soft reboot of your computer, the temporary files are deleted; you must delete these files manually. (Temporary files usually have numbers for names.)

With the memory-cache capabilities of the simulator, your memory map is now restricted only by your PC's capabilities. As a result, there should be sufficient free space on your disk to run any memory map you want to use. If you use the MA command to allocate 20K words (40K bytes) of memory in your memory map, then your disk should have at least 40K bytes of free space available. To do this, you can enter:

```
ma 0x0, 0, 0x5000, ram
```

Note:

You can also use the memory-cache capability feature for the data memory.

5.4 Enabling Memory Mapping



map By default, mapping is enabled when you invoke the debugger. In some instances, you may want to explicitly enable or disable memory. You can use the MAP command to do this; the syntax for this command is:

map on

or **map off**

Note that disabling memory mapping can cause bus fault problems in the target because the debugger may attempt to access nonexistent memory.

Note:

When memory mapping is enabled, you cannot:

- Access memory locations that are not defined by an MA command
- Modify memory areas that are defined as read only or protected

If you attempt to access memory in these situations, the debugger displays this message in the COMMAND window display area:

```
Error in expression
```

5.5 Checking the Memory Map



ml If you want to see which memory ranges are defined, use the ML command. The syntax for this command is:

ml

The ML command lists the page, starting address, ending address, and read/write characteristics of each defined memory range.

For example, if you're using the EVM default memory map and you enter the ML command, the debugger displays this:

<u>Page</u>	<u>Memory range</u>	<u>Attributes</u>
0	0000 - 07ff	READ WRITE
0	0800 - 2bff	READ WRITE
0	2c00 - 7fff	READ WRITE
1	0004 - 0022	READ WRITE
1	0024 - 0026	READ WRITE
1	0028 - 002a	READ WRITE
1	0030 - 0035	READ WRITE
1	0050 - 005f	READ WRITE
1	0060 - 007f	READ WRITE
1	0300 - 04ff	READ WRITE
1	0800 - 2bff	READ WRITE
1	8000 - ffff	READ WRITE

Page 0 = program memory
 Page 1 = data memory

starting address ending address

5.6 Modifying the Memory Map During a Debugging Session



If you need to modify the memory map during a debugging session, use these commands.

md To delete a range of memory from the memory map, use the MD (memory delete) command. The syntax for this command is:

md *address, page*

- address*—identifies the starting address of the range of program, data, or I/O memory. If you supply an *address* that is not the starting address of a range, the debugger displays this error message in the COMMAND window display area:

```
Specified map not found
```

- page*—a one-digit number that identifies the type of memory (program, data, or I/O) that the range occupies:

To identify this page,	Use this value as the <i>page</i> parameter
Program memory	0
Data memory	1
I/O space	2

Note:

If you are using the simulator and want to use the MD command to remove a simulated I/O port, you must first disconnect the port with the MI command. Refer to Section 5.8, *Simulating I/O Space (Simulator Only)*, on page 5-12.

mr If you want to delete all defined memory ranges from the memory map, use the MR (memory reset) command. The syntax for this command is:

mr

This resets the debugger memory map.

ma If you want to add a memory range to the memory map, use the MA (memory add) command. The syntax for this command is:

ma *address, page, length, type*

The MA command is described in detail on page 5-6.

Returning to the original memory map

If you modify the memory map, you may want to go back to the original memory map without quitting and reinvoking the debugger. You can do this by resetting the memory map and then using the TAKE command to read in your original memory map from a batch file.

Suppose, for example, that you had set up your memory map in a batch file named *mem.map*. You could enter these commands to go back to this map:

```
mr  Reset the memory map
take mem.map  Reread the default memory map
```

The MR command resets the memory map. (Note that you could put the MR command in the batch file, preceding the commands that define the memory map.) The TAKE command tells the debugger to execute commands from the specified batch file.

5.7 Using Multiple Memory Maps for Multiple Target Systems

If you're debugging multiple applications, you may need a memory map for each target system. Here's the simplest method for handling this situation.

Step 1: Let the initialization batch file define the memory map for one of your applications.

Step 2: Create a separate batch file that defines the memory map for the additional target system. The filename is unimportant, but for this example, assume that the file is named *filename.x*. The general format of this file's contents should be:

```
mr Reset the memory map
MA commands Define the new memory map
map on Enable mapping
```

(Of course, you can include any other appropriate commands in this batch file.)

Step 3: Invoke the debugger as usual.

Step 4: The debugger reads the initialization batch file during invocation. Before you begin debugging, read in the commands from the new batch file:

```
take filename.x 
```

This redefines the memory map for the current debugging session.

You can also use the `-t` option instead of the TAKE command when you invoke the debugger. The `-t` option allows you to specify a new batch file to be used instead of the default initialization batch file.

5.8 Simulating I/O Space (Simulator Only)

In addition to adding memory ranges to the memory map, you can use the MA command to add I/O ports to the memory map. To do this, use IPORT (input port), OPORT (output port), or IOPORT (input/output port) as the memory type. Use page 1 to simulate serial ports, and use page 2 to simulate parallel ports. Then, you can use the MC command to connect a port to an input or output file. This simulates external I/O cycle reads and writes by allowing you to read data in from a file and/or write data out to a file.

Connecting an I/O port



mc The MC (memory connect) command connects IPORT, OPORT, or IOPORT to an input or output file. The syntax for this command is:

mc *port address, page, filename, fileaccess*

- port address*—defines the address of the I/O port. This parameter is an absolute address, any C expression, the name of a C function, or an assembly-language label.
- page*—a one-digit number that identifies the page that the port occupies. Parallel ports are on page 2 (the I/O space), and serial ports are on page 1 (data space).
- filename*—any filename. If you connect a port to read from a file, the file must exist, or the MC command will fail.
- fileaccess*—identifies the access characteristics of the I/O memory and data memory. Use one of the following keywords to indicate the type of file access you want:
 - **R** or **READ**—Input port file access. Execution continues. You are not notified when the EOF is read. When an EOF is read, the input file rewinds and the simulator resumes reading from the file.
 - **R|NR** or **READ|NR**—Simulator halt at EOF of the input space file access. The simulator halts execution when it reads an EOF. The debugger displays the following message in the COMMAND window:

```
address EOF reached - connected at port:
```

At this point, you can disconnect the file using the ML command and attach a new file using the MC command. If you do nothing, the file rewinds automatically and execution continues until an EOF is read.
 - **W** or **WRITE**—Output port file access.

The file is accessed during an IN or OUT instruction to the associated port address. Any port in I/O space can be connected to a file. A maximum of one input and one output file can be connected to a single port; multiple ports can be connected to a single file. Memory-mapped ports can be connected to files; any instruction that reads from or writes to the memory-mapped port reads from or writes to the associated file. Example 5–2 shows how to connect an input port to an input file.

Example 5–2. Connecting an Input Port to an Input File

Assume that the file in.dat contains words of data in hexadecimal format, one per line, like this:

```
0A00
1000
2000
.
.
.
```

These two debugger instructions set up and connect an input port:

```
MA    0x50,1,0x1,IPOINT      Configure port address 50h
                               as an input port
MC    0x50,1,in.dat,READ     Open file in.dat and
                               connect to port address 50h
```

Assume that these two 'C5x instructions are part of your 'C5x program. They read from the file in.dat.

```
IN    00h,50h               IN instruction reads from file
LAMB  50h                   Memory reference load
                               used to read from file
```

Disconnecting an I/O port

Before you can use the MD command to delete a port from the memory map, you must use the MI command to disconnect the port.



mi The MI (memory disconnect) command disconnects a file from an I/O port. The syntax for this command is:

mi *port address, page, {READ | WRITE}*

The *port address* and *page* identify the port that will be closed. The read/write characteristics must match the parameter used when the port was connected.

5.9 Simulating External Interrupts (Simulator Only)

The 'C5x simulator allows you to simulate the external interrupt signals $\overline{\text{INT1}}$ to $\overline{\text{INT4}}$ and to select the clock cycle where you want an interrupt to occur. To do this, you create a data file and connect it to one of the four interrupt pins, $\overline{\text{INT1}}$ to $\overline{\text{INT4}}$ or the $\overline{\text{BIO}}$ pin.

Note:

The time interval is expressed as a function of CPU clock cycles. Simulation begins at the first clock cycle.

Setting up your input file

In order to simulate interrupts, you must first set up an input file that lists interrupt intervals. Your file must contain a clock cycle in the following format:

`[clock cycle, logic value] rpt {n | EOS}`

Note that the square brackets are used only with logic values for the $\overline{\text{BIO}}$ pin.

- *clock cycle*—represents the CPU clock cycle where you want an interrupt to occur.

You can have two types of CPU clock cycles:

- **Absolute**—To use an absolute clock cycle, your cycle value must represent the actual CPU clock cycle in which you want to simulate an interrupt. For example:

12 34 56

Interrupts are simulated at the 12th, 34th, and 56th CPU clock cycles. No operation is performed on the clock cycle value; the interrupt occurs exactly as the clock cycle value is written.

- **Relative**—You can also select a clock cycle that is relative to the time at which the last event occurred. For example:

12 +34 55

In this example, a total of three interrupts are simulated at the 12th, 46th (12+34), and 55th CPU clock cycles. A plus sign (+) before a clock cycle adds that value to the total clock cycles preceding it. You can mix both relative and absolute values in your input file.

- *logic value*—only for the $\overline{\text{BIO}}$ pin. You must use a value to force the signal to go high or low at the corresponding clock cycle. A value of 1 forces the signal to go high, and a value of 0 forces the signal to go low. For example:

```
[ 12,1] [ 23,0] [ 45,1]
```

This causes the $\overline{\text{BIO}}$ pin to go high at the 12th cycle, low at the 23rd cycle, and high again at the 45th cycle.

- **rpt {n | EOS}**—optional and represents a repetition value.

Two forms of repetition simulate interrupts:

- Repetition on a fixed number of times—You can format your input file to repeat a particular pattern a fixed number of times. For example:

```
5 (+10 +20) rpt 2
```

The values inside of the parentheses represent the portion that is repeated. Therefore, an interrupt is simulated at the 5th CPU cycle, then the 15th (5+10), 35th (15+20), 45th (35+10), and 65th (45+20) CPU clock cycles.

Note that n is a positive integer value.

- Repetition to the end of simulation—To repeat the same pattern throughout the simulation, add the string EOS to the line. For example:

```
10 (+5 +20) rpt EOS
```

Interrupts are simulated at the 10th CPU cycle, then the 15th (10+5), 35th (15+20), 40th (35+5), 60th (40+20), 65th (60+5), and 85th (65+20) CPU cycles, continuing in that pattern until the end of simulation.

Programming the simulator

After creating your input file, you can use debugger commands to connect, list, and disconnect the interrupt pin to your input file. Use these commands as described below, or use them from the PIN pulldown menu.



pinc To connect your input file to the pin, use the following command:

pinc *pinname, filename*

- pinname*—identifies the pin and must be one of four simulated pins ($\overline{\text{INT1}}$ – $\overline{\text{INT4}}$) or the $\overline{\text{BIO}}$ pin.
- filename*—name of your input file.

Example 5–3 shows you how to connect your input file using the PINC command.

Example 5–3. Connecting the Input File With the PINC Command

Suppose you want to generate an $\overline{\text{INT2}}$ external interrupt at the 12th, 34th, 56th, and 89th clock cycles.

First, create a data file with an arbitrary name such as myfile:

```
12 34 56 89
```

Then use the PINC command in the pin pulldown menu to connect the input file to the $\overline{\text{INT2}}$ pin.

```
pinc myfile, int2
```

*Connects your data file
to the specific interrupt pin*

This command connects myfile to the $\overline{\text{INT2}}$ pin. As a result, the simulator generates an $\overline{\text{INT2}}$ external interrupt at the 12th, 34th, 56th, and 89th clock cycles.

pinl To verify that your input file is connected to the correct pin, use the PINL command. The syntax for this command is:

pinl

The PINL command displays all of the unconnected pins first, followed by the connected pins. For a pin that has been connected, it displays the name of the pin and the absolute pathname of the file in the COMMAND window.

```

COMMAND
PIN          FILENAME
-----
INT1         NULL
INT3         NULL
INT4         NULL
BIO          NULL
- INT2       /320h11/myfile
>>>
  
```

pinl To end the interrupt simulation, disconnect the pin. You can do this with the following command:

pinl *pinname*

The *pinname* parameter identifies the interrupt pin and must be one of the external interrupt pins ($\overline{\text{INT1}}$ – $\overline{\text{INT4}}$) or the $\overline{\text{BIO}}$ pin. The PIND command detaches the file from the input pin. After executing this command, you can connect another file to the same pin.

5.10 Simulating Peripherals (Simulator Only)

With the 'C5x simulator, you can simulate the timer, standard serial port, buffered serial port, and TDM serial port. The peripherals simulated depend upon the device that you simulate. You simulate a device by starting the simulator (sim5x command) with the appropriate option. Table 5–1 summarizes the options used to simulate the peripherals for each device.

Table 5–1. Debugger Options for the Simulator

Option	Device Simulated	Peripherals Simulated
–mv50	'C50	Serial port, TDM serial port
–mv51	'C51	Serial port, TDM serial port
–mv53	'C53	Serial port, TDM serial port
–mv56	'C56	Serial port, buffered serial port
–mv57	'C57	Serial port, buffered serial port

Detailed information about simulating the different types of serial ports is discussed in the following sections:

Type of Serial Port	See Section . . .
Standard	5.11 on page 5-19
Buffered	5.12 on page 5-22
TDM	5.13 on page 5-25

5.11 Simulating Standard Serial Ports (Simulator Only)

The 'C5x simulator supports standard serial port transmission and reception by reading data from, and writing data to, the files associated with the DXR/TDXR and DRR/TRCV registers, respectively.

The simulator also provides limited support for the simulation of the serial port control pins (frame synchronization pins) with the help of external event simulation capability. Frame synchronization pin values for receive and transmit operations at various instants of time are fed through the files associated with the pins.

The 'C5x simulator supports the following operations in the standard serial port simulation:

- Internal clocks (1/4 CPU clock) and external clocks for the transmit and receive operations. External clocks are simulated by using the DIVIDE command (described on page 5-20) in the files connected to the FSX/TFSX and FSR/TFSR pins.
- External frame synchronization pulses (FSX/TFSX transmit and FSR/TFSR receive frame synchronization pulses). Transmit and receive operations are initiated when the signals for these values go high.
- The operations associated with the following memory-mapped registers:

Register	Memory	Bits Used	Description
SPC	0x22	FO	Format specifier (8/16 bits)
TSPC	0x32	MCM	Internal/external clock
		XRST/RRST	Transmit/receive reset
		XRDY/RRDY	Transmit/receive ready
		XSREMPY	Transmit register empty flag
		RSRFULL	Receive register full flag
DXR	0x21	All bits are used	Transmit data register
TDXR	0x31		
DRR	0x20	All bits are used	Receive data register
TRCV	0x30		

Setting up your transmit and receive operations

The 'C5x simulator supports the simulation of the following pins using external event simulation. The pulses occurring on the FSX and FSR pins initiate the standard serial port transmit and receive operations, respectively.

- FSR/TFSR—Frame synchronization pulses for the receive operation
- FSX/TFSX—Frame synchronization pulses for the transmit operation

Connect the files to the pins using the PINC (pin connect) command (described on page 5-16). Use the following command syntax, selecting the appropriate command for the pin you want:

```
pinc FSX, filename  
pinc TFSX, filename  
pinc FSR, filename  
pinc TFSR, filename
```

filename is the name of the file that contains the CPU cycles at which the pin value goes high. Use the following syntax in the files to define clock cycles:

```
[clock cycle] rpt {n | EOS}
```

Note that the square brackets are used only with logic values for the $\overline{\text{BIO}}$ pin. For more information about defining clock cycles, see Section 5.9, *Simulating External Interrupts (Simulator Only)*, on page 5-14.

Additionally, you can use the DIVIDE command to specify the clock divide ratio for the device. Use the following syntax in the files for the DIVIDE command:

```
DIVIDE r
```

r is a real number or integer specifying the ratio of serial port clock versus the CPU clock. Use the divide ratio when the serial port is configured to use the external clock. When you use the DIVIDE command, it must be the first command in the file.

The following example specifies the clock ratio of the transmit clock and the clock cycles for the occurrence of TFSX pulses (if this file is connected to the TFSX pin):

```
DIVIDE 5  
100 +200 +100
```

The DIVIDE command specifies the divide-down ratio of the clock against the CPU clock. That is, the CLKX frequency is 1/5 of the CPU clock. The second line indicates that the TFSX should go high at the 100th, 300th (100 + 200), and 400th (300 + 100) CPU cycles. The TFSX pin goes high in the 500th, 1500th, and 2000th cycles of the serial port clock.

Connecting input/output files

Input and output files are connected to DRR/TRCV and DXR/TDXR registers for receive and transmit operations, respectively. To simulate the transmit operation, data is written to the file that is connected to the DXR/TDXR register. To simulate the receive operation, data is read from the file that is connected to the DRR/TRCV register. The input and output file formats for the standard serial port operation requires at least one line containing a hexadecimal number. The following is an acceptable format for an input file:

```
0055
aa55
efef
dead
```

Note:

To simulate the standard serial port 0, use the DXR and DRR registers and the FSX and FSR pins. To simulate the standard serial port 1, use the TDXR and TRCV registers and the TFSX and TFSR pins.

Programming the simulator

To simulate the standard serial port, configure the DXR/TDXR and DRR/TRCV registers as the output port (OPOINT) and the input port (IPOINT), respectively. Connect these ports to an output file and an input file. Also, connect files to the TFSX/FSX and TFSR/FSR pins to specify the clock cycles during which the frame synchronization pins go high. To make these connections, use the following commands in the simulator initialization batch file (siminit.cmd):

```
ma DRR,1,1,IPOINT
ma DXR,1,1,OPOINT

mc DRR,1,1,receive filename,READ
mc DXR,1,1,transmit filename,WRITE

pinc FSX,fsx timing filename
pinc FSR,fsr timing filename
```

Variable	Description
<i>receive filename</i>	The file to read data from, which simulates the input port
<i>transmit filename</i>	The file to write data to, which simulates the output port
<i>fsx timing filename</i>	The file that contains the CPU cycles at which the FSX frame synchronization pin goes high
<i>fsr timing filename</i>	The file that contains the CPU cycles at which the FSR frame synchronization pin goes high

5.12 Simulating Buffered Serial Ports (Simulator Only)

The 'C5x simulator supports buffered serial port transmission and reception by reading data from and writing data to the files associated with the DXR and DRR registers, respectively. The simulator provides limited support for the serial port control pins (frame synchronization pins) using external event simulation capability. Pin values for receive and transmit operations at various instants of time are fed through the files associated with the pins. The simulator supports the following operations in the buffered serial port simulation:

- Automatic buffering and standard serial port modes
- Internal clocks ($1/(\text{CLKDV} + 1)$ CPU clock) and external clocks for the transmit and receive operations
- External frame synchronization pulses (FSX transmit and FSR receive frame synchronization pulses). Transmit and receive operations are initiated when the signals for these values go high.
- The operations associated with the following memory-mapped registers:

Register	Memory	Bits Used	Description
SPC	0x22	FO	Format specifier (8/16 bits)
		MCM	Internal/external clock
		XRST/RRST	Transmit/receive reset
		XRDY/RRDY	Transmit/receive ready
		XSREMPY	Transmit register empty flag
		RSRFULL	Receive register full flag
DXR	0x31	All bits are used	Transmit data register
DRR	0x30	All bits are used	Receive data register
SPCE	0x33	CLKDV	Clock divide ratio
		FE	Extended format specifier
		RH/TH	Buffer half received or transmitted
		BXE/BRE	Enable/disable automatic buffering
		HALTX/HALTR	Switch to standalone mode after the current half is transmitted/received
AXR	0x34	All bits are used	Address register for transmit
ARR	0x36	All bits are used	Address register for receive
BKX	0x35	All bits are used	Block size register for the transmit
BKR	0x37	All bits are used	Block size register for the receive

Note:

In the simulator, the address of the buffered serial port registers and the TDM serial port registers are the same. You can simulate one or the other but not both at the same time.

Setting up your transmit and receive operations

The 'C5x simulator supports the simulation of the following pins using external event simulation. The pulses occurring on the FSX and FSR pins initiate the buffered serial port transmit and receive operations, respectively.

- FSR—Frame synchronization pulses for the receive operation
- FSX—Frame synchronization pulses for the transmit operation

Connect the files to the pins using the PINC (pin connect) command (described on page 5-16). Use the following command syntax, selecting the appropriate command for the pin you want:

pinc FSX, *filename*

pinc FSR, *filename*

filename is the name of the file that contains the CPU cycles at which the pin value goes high. Use the following syntax in the files to define clock cycles:

[clock cycle] **rpt** {*n* | **EOS**}

Note that the square brackets are used only with logic values for the $\overline{\text{BIO}}$ pin. For more information about defining clock cycles, see Section 5.9, *Simulating External Interrupts (Simulator Only)*, on page 5-14.

Additionally, you can use the DIVIDE command to specify the clock divide ratio for the device. Use the following syntax in the files for the DIVIDE command:

DIVIDE *r*

r is a real number or integer specifying the ratio of serial port clock versus the CPU clock. Use the divide ratio when the serial port is configured to use the external clock. When you use the DIVIDE command, it must be the first command in the file.

The following example specifies the clock ratio of the transmit clock and the clock cycles for the occurrence of TFSX pulses (if this file is connected to the TFSX pin):

```
DIVIDE 5
100 +200 +100
```

The DIVIDE command specifies the divide-down ratio of the clock against the CPU clock. That is, the CLKX frequency is 1/5 of the CPU clock. The second line indicates that the TFSX should go high at the 100th, 300th (100 + 200), and 400th (300 + 100) CPU cycles. The TFSX pin goes high in the 500th, 1500th, and 2000th cycles of the serial port clock.

Connecting input/output files

Input and output files are connected to DRR and DXR registers for receive and transmit operations respectively. To simulate the transmit operation, data is written to the file that is connected to the DXR register. To simulate the receive operation, data is read from the file that is connected to the DRR register.

The input and output file formats for the buffered serial port operation requires at least one line containing a hexadecimal number. The following example shows an acceptable format for an input file:

```
0055
aa55
efef
dead
```

Programming the simulator

To simulate the buffered serial port, configure the DXR and DRR registers as the output port (OPORT) and the input port (IPORT), respectively. Connect these ports to an output file and an input file. Also, connect files to the TFSX/FSX and TFSR/FSR pins to specify the clock cycles during which the frame synchronization pins go high.

To make these connections, use the following commands in the simulator initialization batch file (siminit.cmd):

```
ma DRR,1,1,IPORT
ma DXR,1,1,OPORT

mc DRR,1,1,receive filename,READ
mc DXR,1,1,transmit filename,WRITE

pinc FSX,fsx timing filename
pinc FSR,fsr timing filename
```

Variable	Description
<i>receive filename</i>	The file to read data from, which simulates the input port
<i>transmit filename</i>	The file to write data to, which simulates the output port
<i>fsx timing filename</i>	The file that contains the CPU cycles at which the FSX frame synchronization pin goes high
<i>fsr timing filename</i>	The file that contains the CPU cycles at which the FSR frame synchronization pin goes high

5.13 Simulating TDM Serial Ports (Simulator Only)

The 'C5x simulator supports TDM serial port transmission and reception by reading data from and writing data to the files associated with the TDXR and TRCV registers, respectively.

The simulator also provides limited support for the simulation of the TDM port control pins (frame synchronization pins) with the help of external event simulation capability. Frame synchronization pin values for receive and transmit operations at various instants of time are fed through the files associated with the pins.

The 'C5x simulator supports the following operations in the TDM serial port simulation:

- TDM and standard serial port modes
- Internal clocks (1/4 CPU clock) and external clocks for the transmit and receive operations. External clocks are simulated by using the DIVIDE command in the files connected to the TFSX and TFSR pins.
- External frame synchronization pulses (TFSX transmit and TFSR receive frame synchronization pulses). Transmit and receive operations are initiated when the signals for these values go high.
- The operations associated with the following memory-mapped registers:

Register	Memory	Bits Used	Description
TSPC	0x32	TDM	Multiprocessor/normal mode
		MCM	Internal/external clock
		XRST/RRST	Transmit/receive reset
		XRDY/RRDY	Transmit/receive ready
		XSREMPY	Transmit register empty flag
		RSRFULL	Receive register full flag
TCSR	0x33	All bits are used	Channel select register
TRTA	0x34	All bits are used	Receive/transmit address register
TRAD	0x35	All bits are used	Receive address register
TDXR	0x31	All bits are used	Transmit data register
TRCV	0x30	All bits are used	Receive data register

Setting up your transmit and receive operations

The 'C5x simulator supports the simulation of the following pins using external event simulation. The pulses occurring on the TFSX and TFSR pins initiate the TDM serial port transmit and receive operations, respectively.

- TFSR—Frame synchronization pulses for the receive operation
- TFSX—Frame synchronization pulses for the transmit operation

Connect the files to the pins using the PINC (pin connect) command (described on page 5-16). Use the following command syntax, selecting the appropriate command for the pin you want:

pinc TFSX, filename
pinc TFSR, filename

filename is the name of the file that contains the CPU cycles at which the pin value goes high. Use the following syntax in the files to define clock cycles:

`[clock cycle] rpt {n | EOS}`

Note that the square brackets are used only with logic values for the $\overline{\text{BIO}}$ pin. For more information about defining clock cycles, see Section 5.9, *Simulating External Interrupts (Simulator Only)*, on page 5-14.

Additionally, you can use the DIVIDE command to specify the clock divide ratio for the device. Use the following syntax in the files for the DIVIDE command:

DIVIDE r

r is a real number or integer specifying the ratio of serial port clock versus the CPU clock. Use the divide ratio when the serial port is configured to use the external clock. When you use the DIVIDE command, it must be the first command in the file.

The following example specifies the clock ratio of the transmit clock and the clock cycles for the occurrence of TFSX pulses (if this file is connected to the TFSX pin):

```
DIVIDE 5  
100 +200 +100
```

The DIVIDE command specifies the divide-down ratio of the clock against the CPU clock. That is, the CLKX frequency is 1/5 of the CPU clock. The second line indicates that the TFSX should go high at the 100th, 300th (100 + 200), and 400th (300 + 100) CPU cycles. The TFSX pin goes high in the 500th, 1500th, and 2000th cycles of the serial port clock.

Connecting input/output files

Input and output files are connected to TRCV and TDXR registers for receive and transmit operations, respectively. To simulate the transmit operation, data is written to the file that is connected to the TDXR register. To simulate the receive operation, data is read from the file that is connected to the TRCV register. Use the following syntax to create the files:

channel-address data

channel-address specifies the TDM channel in which transmission/reception takes place. *data* specifies the value that is written or read from the file. Each field is in hexadecimal format separated by spaces. The following is an acceptable format for an input file:

```
10 0055
34 aa55
80 efef
01 dead
```

Programming the simulator

To simulate the TDM serial port, configure the TDXR and TRCV registers as the output port (OPOINT) and the input port (IPOINT), respectively. Connect these ports to an output file and an input file. Also, connect files to the TFSX/FSX and TFSR/FSR registers to specify the clock cycles during which the frame synchronization pins go high.

To make these connections, use the following commands in the simulator initialization batch file (siminit.cmd):

```
ma TRCV,1,1,IPOINT
ma TDXR,1,1,OPOINT

mc TRCV,1,1,receive filename,READ
mc TDXR,1,1,transmit filename,WRITE

pinc TFSX,fsx timing filename
pinc TFSR,fsr timing filename
```

Variable	Description
<i>receive filename</i>	The file to read data from, which simulates the input port
<i>transmit filename</i>	The file to write data to, which simulates the output port
<i>fsx timing filename</i>	The file that contains the CPU cycles at which the FSX frame synchronization pin goes high
<i>fsr timing filename</i>	The file that contains the CPU cycles at which the FSR frame synchronization pin goes high

Index

A

- addresses
 - accessible locations 5-2
 - I/O address space
 - simulator 5-10, 5-12 to 5-13
 - invalid memory 5-3
 - nonexistent memory locations 5-2
 - protected areas 5-3, 5-8
 - undefined areas 5-3, 5-8
- arrow keys
 - HP systems 3-9
 - SPARC systems 2-9
- assembler
 - HP systems 3-3
 - PC systems 1-3
 - SPARC systems 2-3
- autoexec.bat file
 - environment variables 1-5
 - sample 1-7

B

- b debugger option
 - HP systems 3-6
 - PC systems 1-5
 - SPARC systems 2-6
- batch files
 - autoexec.bat 1-7
 - .cshrc
 - HP systems 3-6 to 3-7
 - SPARC systems 2-6 to 2-7
 - emuinit.cmd 5-11
 - init.clr
 - HP systems 3-3
 - PC systems 1-3
 - SPARC systems 2-3
 - init.cmd 5-2

- batch files (continued)
 - initdb.bat 1-6
 - initialization 5-2, 5-4
 - mem.map 5-11
 - memory maps 5-11
 - mono.clr
 - HP systems 3-3
 - PC systems 1-3
 - SPARC systems 2-3
 - simit.cmd
 - HP systems 3-3
 - PC systems 1-3
 - SPARC systems 2-3
 - TAKE command 5-11
- bb debugger option
 - HP systems 3-6
 - PC systems 1-5
 - SPARC systems 2-6
- BIO pseudoregister 5-14 to 5-17
- BLPD instruction 4-2
- BPT pseudoinstruction 4-2

C

- CD-ROM
 - mounting
 - HP systems 3-4
 - SPARC systems 2-4
 - requirements
 - HP systems 3-2
 - SPARC systems 2-2
 - retrieving files from
 - HP systems 3-4
 - SPARC systems 2-5
 - unmounting
 - HP systems 3-5
 - SPARC systems 2-5
- COFF files 5-3

- color mapping
 - HP systems 3-10
 - SPARC systems 2-10
- commands
 - memory commands 5-6 to 5-27
- compiler
 - HP systems 3-3
 - PC systems 1-3
 - SPARC systems 2-3
- contacting TI vi
- .cshrc file
 - contents
 - HP systems 3-6 to 3-7
 - SPARC systems 2-6 to 2-7
 - invoking
 - HP systems 3-7
 - SPARC systems 2-7
 - sample
 - HP systems 3-7
 - SPARC systems 2-7
- customizing the display
 - HP systems 3-3, 3-10
 - PC systems 1-2, 1-3
 - SPARC systems 2-3, 2-10

D

- D_DIR environment variable
 - HP systems 3-6
 - PC systems 1-5
 - SPARC systems 2-6
- D_OPTIONS environment variable
 - HP systems 3-6
 - PC systems 1-5
 - SPARC systems 2-6
- D_SRC environment variable
 - HP systems 3-6
 - PC systems 1-5
 - SPARC systems 2-6
- data memory
 - adding to memory map 5-6
 - deleting from memory map 5-10
- debugger
 - environment setup
 - HP systems 3-6 to 3-7
 - PC systems 1-5 to 1-7
 - SPARC systems 2-6 to 2-7

- debugger (continued)
 - font changes
 - HP systems 3-10
 - SPARC systems 2-10
 - installation
 - procedure
 - HP systems 3-4
 - PC systems 1-4
 - SPARC systems 2-4
 - verification
 - HP systems 3-8
 - PC systems 1-8
 - SPARC systems 2-8
 - Windows 1-4, 1-9
 - X Window System
 - HP systems 3-9 to 3-10
 - SPARC systems 2-9 to 2-10
- default
 - memory map 5-4
 - HP systems 3-3
 - PC systems 1-3
 - SPARC systems 2-3
 - screen configuration file
 - HP systems 3-3
 - PC systems 1-3
 - SPARC systems 2-3
- directories
 - auxiliary files
 - HP systems 3-6
 - PC systems 1-5
 - SPARC systems 2-6
 - debugger software
 - HP systems 3-4, 3-6
 - PC systems 1-4, 1-5
 - SPARC systems 2-5, 2-6
 - identifying additional source directories
 - HP systems 3-6
 - PC systems 1-5
 - SPARC systems 2-6
 - sim5x
 - HP systems 3-4, 3-6
 - PC systems 1-4, 1-5
 - SPARC systems 2-5, 2-6
- disk space requirements
 - HP systems 3-2
 - SPARC systems 2-2

display
 font changes
 HP systems 3-10
 SPARC systems 2-10
 requirements
 HP systems 3-2
 PC systems 1-2
 SPARC systems 2-2
 DISPLAY environment variable
 HP systems 3-6
 SPARC systems 2-6

E

end key
 HP systems 3-9
 SPARC systems 2-9
 environment variables
 HP systems 3-6
 PC systems 1-5
 SPARC systems 2-6
 external interrupts
 connect input file 5-16
 disconnect pins 5-17
 list pins 5-17
 PINC command 5-16
 PIND command 5-17
 PINL command 5-17
 programming simulator 5-16
 setting up input file
 absolute clock cycle 5-14
 relative clock cycle 5-14
 repetition 5-15
 simulating 5-14

F

files
 connecting to
 buffered serial ports 5-24
 I/O ports 5-12
 standard serial ports 5-21
 TDM serial ports 5-27
 disconnecting from I/O ports 5-13
 font changes
 HP systems 3-10
 SPARC systems 2-10
 -font debugger option 1-5

frame synchronization pins
 buffered serial port 5-23
 standard serial port 5-20
 TDM serial port 5-26
 function key mapping
 HP systems 3-9
 SPARC systems 2-9

G

graphics card requirements 1-2

H

hardware checklist
 HP systems 3-2
 PC systems 1-2
 SPARC systems 2-2
 home key
 HP systems 3-9
 SPARC systems 2-9
 host
 HP systems 3-2
 PC systems 1-2
 SPARC systems 2-2

I

-i debugger option
 HP systems 3-6
 PC systems 1-5
 SPARC systems 2-6
 I/O memory
 adding to memory map 5-6
 deleting from memory map 5-10
 simulating 5-10, 5-12 to 5-13
 init.clr file
 HP systems 3-3
 PC systems 1-3
 SPARC systems 2-3
 init.cmd file 5-2
 initdb.bat file
 sample 1-6
 initialization batch files 5-2, 5-4
 HP systems 3-3
 init.cmd 5-2
 PC systems 1-3
 SPARC systems 2-3

- insert key
 - HP systems 3-9
 - SPARC systems 2-9
- installation
 - debugger
 - HP systems* 3-4
 - PC systems* 1-4
 - SPARC systems* 2-4
 - simulator
 - HP systems* 3-4
 - PC systems* 1-4
 - SPARC systems* 2-4
 - verifying
 - HP systems* 3-8
 - PC systems* 1-8
 - SPARC systems* 2-8
- interrupt pins 5-14
- invalid memory addresses 5-3, 5-8

K

- keyboard mapping
 - HP systems 3-9
 - SPARC systems 2-9
- keysym label
 - HP systems 3-9
 - SPARC systems 2-9

L

- linker
 - command files
 - MEMORY definition 5-2
 - HP systems 3-3
 - PC systems 1-3
 - SPARC systems 2-3

M

- MA command 5-4, 5-6, 5-10
- machine to display on
 - HP systems 3-6
 - SPARC systems 2-6
- MAP command 5-8
- mapping keys
 - HP systems 3-9
 - SPARC systems 2-9
- MC command 5-12

- MD command 5-10
- memory
 - batch file search order 5-2
 - default map 5-4
 - HP systems* 3-3
 - PC systems* 1-3
 - SPARC systems* 2-3
 - invalid addresses 5-3
 - invalid locations 5-8
 - map
 - defining* 5-2
 - in a batch file 5-2
 - interactively 5-2
 - modifying* 5-2
 - potential problems* 5-3
 - mapping
 - adding ranges* 5-6
 - commands*
 - MA command 5-4, 5-6, 5-10
 - MAP command 5-8
 - MD command 5-10
 - ML command 5-9
 - MR command 5-10
 - deleting ranges* 5-10
 - disabling* 5-8
 - HP systems* 3-3
 - listing current map* 5-9
 - modifying* 5-10
 - multiple maps* 5-11
 - PC systems* 1-3
 - resetting* 5-10
 - returning to default* 5-11
 - simulating I/O ports* 5-12, 5-13
 - SPARC systems* 2-3
 - nonexistent locations 5-2
 - protected areas 5-3, 5-8
 - requirements 1-2
 - simulating
 - I/O memory* 5-10, 5-12 to 5-13
 - ports*
 - MC command 5-12
 - MI command 5-13
 - undefined areas 5-3, 5-8
 - valid types 5-6
- MI command 5-13
- ML command 5-9
- modifying
 - memory map 5-2, 5-10

mono.clr file
 HP systems 3-3
 PC systems 1-3
 SPARC systems 2-3

monochrome monitors color mapping
 HP systems 3-10
 SPARC systems 2-10

mouse requirements
 HP systems 3-2
 PC systems 1-2
 SPARC systems 2-2

MR command 5-10

-mv debugger option
 PC systems 1-5
 SPARC systems 2-6

N

nonexistent memory locations 5-2

O

operating system
 HP systems 3-3
 PC systems 1-3
 SPARC systems 2-3

optional files
 HP systems 3-3
 PC systems 1-3
 SPARC systems 2-3

P

-p debugger option 1-5

page-down key
 HP systems 3-9
 SPARC systems 2-9

page-up key
 HP systems 3-9
 SPARC systems 2-9

PATH statement 1-5

peripherals
 simulating 5-18

permissions
 HP systems 3-3
 SPARC systems 2-3

PINC command 5-16

PIND command 5-17

PINL command 5-17

port address
 simulator 5-10, 5-12 to 5-13

ports
 simulating 5-12 to 5-13

-profile debugger option
 HP systems 3-6
 PC systems 1-5
 SPARC systems 2-6

program memory
 adding to memory map 5-6
 deleting from memory map 5-10

R

receive operation
 buffered serial port simulation 5-23
 standard serial port simulation 5-20
 TDM serial port simulation 5-26

retrieving files from CD-ROM
 HP systems 3-4
 SPARC systems 2-5

root privileges
 HP systems 3-3
 SPARC systems 2-3

S

-s debugger option
 HP systems 3-6
 PC systems 1-5
 SPARC systems 2-6

screen configurations
 HP systems 3-3
 PC systems 1-3
 SPARC systems 2-3

serial port simulation
 buffered 5-22 to 5-24
 standard 5-19 to 5-21
 TDM 5-25 to 5-28

serial ports
 buffered
 programming simulator 5-24
 simulation 5-12 to 5-13
 standard
 programming simulator 5-21
 TDM
 programming simulator 5-27

setpath statement
 HP systems 3-6
 SPARC systems 2-6

sim5x
 command options
 HP systems 3-6
 PC systems 1-5
 SPARC systems 2-6
 directory
 HP systems 3-4, 3-6
 PC systems 1-4, 1-5
 SPARC systems 2-5, 2-6
 verifying the installation
 HP systems 3-8
 PC systems 1-8
 SPARC systems 2-8

siminit.cmd file
 HP systems 3-3
 PC systems 1-3
 SPARC systems 2-3

simulating
 buffered serial ports 5-22 to 5-24
 interrupts 5-14
 peripherals 5-18
 standard serial ports 5-19 to 5-21
 TDM serial ports 5-25 to 5-28

simulator
 environment setup
 HP systems 3-6 to 3-7
 PC systems 1-5 to 1-7
 SPARC systems 2-6 to 2-7
 I/O memory 5-10, 5-12 to 5-13
 installation
 procedure
 HP systems 3-4
 PC systems 1-4
 SPARC systems 2-4
 verification
 HP systems 3-8
 PC systems 1-8
 SPARC systems 2-8

simulator programming
 buffered serial ports 5-24
 standard serial ports 5-21
 TDM serial ports 5-27

software
 checklist
 HP systems 3-3
 PC systems 1-3
 SPARC systems 2-3
 tools
 HP systems 3-3
 PC systems 1-3
 SPARC systems 2-3

special keys
 HP systems 3-9
 SPARC systems 2-9

T

-t debugger option
 during debugger invocation 5-2
 HP systems 3-6
 PC systems 1-5
 SPARC systems 2-6

TAKE command 5-11

TBLR instruction 4-2

transmit operation
 buffered serial port simulation 5-23
 standard serial port simulation 5-20
 TDM serial port simulation 5-26

U

utilities
 xev
 HP systems 3-9
 SPARC systems 2-9
 xmodmap
 HP systems 3-9
 SPARC systems 2-9
 xrdb
 HP systems 3-10
 SPARC systems 2-10

V

-v debugger option
 HP systems 3-6
 PC systems 1-5
 SPARC systems 2-6

verifying the installation

- HP systems 3-8
- PC systems 1-8
- SPARC systems 2-8

W

Windows 1-9

X

-x debugger option

- HP systems 3-6
- PC systems 1-5
- SPARC systems 2-6

X Window System

- HP systems 3-9 to 3-10
- SPARC systems 2-9 to 2-10

.Xdefaults file

- HP systems 3-10
- SPARC systems 2-10

xev utility

- HP systems 3-9
- SPARC systems 2-9

xmodmap utility

- HP systems 3-9
- SPARC systems 2-9

xrdb utility

- HP systems 3-10
- SPARC systems 2-10

