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.

Copyright © 1996, Texas Instruments Incorporated
About This Manual

The TMS320C54x Simulator Getting Started Guide tells you how to install the TMS320C54x simulator debugging tools on your system. It also gives you the following information:

- How to set environment variables for parameters that you use often
- How to verify the software installation
- How to define and use a memory map for the TMS320C54x to simulate ports

How to Use This Manual

The goal of this book is to get you started using the simulator specifically designed for the TMS320C54x. Following are the topics covered in this getting started guide:

<table>
<thead>
<tr>
<th>For information about …</th>
<th>See …</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting up the debugger on a PC™ running Windows™ 3.1: installing the simulator and debugger software, setting environment variables, and verifying the installation</td>
<td>Chapter 1</td>
</tr>
<tr>
<td>Setting up the debugger on a SPARCstation™ running SunOS™: installing the simulator and debugger software, setting environment variables, and verifying the installation</td>
<td>Chapter 2</td>
</tr>
<tr>
<td>Setting up the debugger on a HP™ workstation running HP-UX™: installing the simulator and debugger software, setting environment variables, and verifying the installation</td>
<td>Chapter 3</td>
</tr>
<tr>
<td>Release notes and enhancements</td>
<td>Chapter 4</td>
</tr>
<tr>
<td>Defining and using a memory map for the TMS320C54x to simulate ports</td>
<td>Chapter 5</td>
</tr>
</tbody>
</table>
Notational Conventions

The abbreviation 'C54x refers to any and all TMS320C54x devices except where individually noted. The devices are:

- TMS320C541
- TMS320C542
- TMS320C543
- TMS320C545
- TMS320C546
- TMS320C548
- TMS320C545LP
- TMS320LC541
- TMS320LC542
- TMS320LC543
- TMS320LC545
- TMS320LC546
- TMS320VC541
- TMS320VC542
- TMS320VC543
- TMS320VC545
- TMS320VC546
- TMS320VC548

Program listings, program examples, and interactive displays are shown in a special typeface. 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 an example of a command that you might enter:

```
  cd /cdrom/hp
```

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 command syntax:

```
  wd  index number [, window name]
```

`wd` is the command. This command has two parameters, `index number` and `window name`.

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 a command that has an optional parameter:

```
  emurst [options]
```

This command allows you to specify one or more options.

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

```
  map { on | off }
```

This provides two choices: `map on` or `map off`.

Unless the list is enclosed in square brackets, you must choose one item from the list.
Related Documentation From Texas Instruments

The following books describe the TMS320C54x devices 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.

**TMS320C5xx C Source Debugger User’s Guide** (literature number SPRU099) tells you how to invoke the 'C54x 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. It also includes a tutorial that introduces basic debugger functionality.

**TMS320C54x Assembly Language Tools User’s Guide** (literature number SPRU102) 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 'C54x generation of devices.

**TMS320C54x Optimizing C Compiler User’s Guide** (literature number SPRU103) describes the 'C54x C compiler. This C compiler accepts ANSI standard C source code and produces TMS320 assembly language source code for the 'C54x generation of devices.
Related Documentation From Texas Instruments

**TMS320C54x DSP Reference Set** is composed of four volumes that can be ordered as a set with literature number SPRU210. To order an individual book, use the document-specific literature number:

**TMS320C54x DSP Reference Set, Volume 1: CPU and Peripherals**
(literature number SPRU131) describes the TMS320C54x 16-bit, fixed-point, general-purpose digital signal processors. Covered are its architecture, internal register structure, data and program addressing, the instruction pipeline, DMA, and on-chip peripherals. Also includes development support information, parts lists, and design considerations for using the XDS510 emulator.

**TMS320C54x DSP Reference Set, Volume 2: Mnemonic Instruction Set**
(literature number SPRU172) describes the TMS320C54x digital signal processor mnemonic instructions individually. Also includes a summary of instruction set classes and cycles.

**TMS320C54x DSP Reference Set, Volume 3: Algebraic Instruction Set**
(literature number SPRU179) describes the TMS320C54x digital signal processor algebraic instructions individually. Also includes a summary of instruction set classes and cycles.

**TMS320C54x DSP Reference Set, Volume 4: Applications Guide**
(literature number SPRU173) describes software and hardware applications for the TMS320C54x digital signal processor. Also includes development support information, parts lists, and design considerations for using the XDS510 emulator.
Trademarks

320 Hotline On-line is a trademark of Texas Instruments Incorporated.

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

IBM, PC, PC/AT, and PC-DOS are trademarks of International Business Machines Corp.

Microsoft, MS-DOS, and Windows are registered trademarks of Microsoft Corporation.

OpenWindows, SunOS, Solaris, Sun Type 4, and Sun Type 5 are trademarks of Sun Microsystems, Inc.

SPARC and SPARCstation are trademarks of SPARC International, Inc. and are licensed exclusively to Sun Microsystems, Inc.

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

X Window System is a trademark of the Massachusetts Institute of Technology.
If You Need Assistance

If You Need Assistance . . .

- **World-Wide Web Sites**
  - TI Online http://www.ti.com
  - Semiconductor Product Information Center (PIC) http://www.ti.com/sc/docs/pic/home.htm
  - DSP Solutions http://www.ti.com/dsp
  - 320 Hotline On-line http://www.ti.com/sc/docs/dsp/support.html

- **North America, South America, Central America**
  - Product Information Center (PIC) (972) 644-5580
  - TI Literature Response Center U.S.A. (800) 477-8924
  - Software Registration/Upgrades (214) 638-0333 Fax: (214) 638-7742
  - U.S. Factory Repair/Hardware Upgrades (281) 274-2285
  - U.S. Technical Training Organization (972) 644-5580
  - DSP Hotline (281) 274-2320 Fax: (281) 274-2324 Email: dsph@ti.com
  - DSP Modem BBS (281) 274-2323
  - DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/mirrors/tms320bbs

- **Europe, Middle East, Africa**
  - European Product Information Center (EPIC) Hotlines:
    - Multi-Language Support +33 1 30 70 11 69 Fax: +33 1 30 70 10 32 Email: epic@ti.com
    - Deutsch +49 8161 80 33 11 or +33 1 30 70 11 68
    - English +33 1 30 70 11 65
    - Français +33 1 30 70 11 64
    - Italiano +33 1 30 70 11 67
    - EPIC Modem BBS +33 1 30 70 11 99
    - European Factory Repair +33 4 93 22 25 40
    - Europe Customer Training Helpline Fax: +49 81 61 80 40 10

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

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

- **Documentation**
  When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.
  Mail: Texas Instruments Incorporated Technical Documentation Services, MS 702 P.O. Box 1443 Houston, Texas 77251-1443
  Email: comments@books.sc.ti.com

**Note:** When calling a Literature Response Center to order documentation, please specify the literature number of the book.
1 Installing the Simulator and C Source Debugger With Windows 3.1 1-1
Lists the hardware and software you need to install the simulator and C source debugger; provides installation instructions for PC systems running Windows 3.1.

1.1 System Requirements 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
Modifying the PATH statement 1-6
Setting up the environment variables 1-6
Invoking the modified or new batch file 1-7

1.4 Step 3: Verifying the Installation 1-8

2 Installing the Simulator and C Source Debugger With SunOS 2-1
Lists the hardware and software you need to install the simulator and C source debugger; provides installation instructions for SPARCstations running SunOS.

2.1 System Requirements 2-2
Hardware checklist 2-2
Software checklist 2-2

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
Modifying the path statement 2-6
Setting up the environment variables 2-6
Reinitializing your shell 2-8

2.4 Step 3: Verifying the Installation 2-9

2.5 Using the Debugger With the X Window System 2-10
Using the special keys on the keyboard 2-10
Changing the debugger font 2-11
Color mappings on monochrome screens 2-11
3 Installing the Simulator and C Source Debugger With HP-UX .............................................. 3-1
Lists the hardware and software you need to install the simulator and C source debugger; provides installation instructions for HP workstations running HP-UX.

3.1 System Requirements ................................................................. 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 and setting up the simulator ................................. 3-4
Unmounting the CD-ROM ................................................................ 3-5

3.3 Step 2: Setting Up the Debugger Environment .......................... 3-6
Modifying the path statement ......................................................... 3-6
Setting up the environment variables ............................................. 3-6
Reinitializing your shell .................................................................. 3-8

3.4 Step 3: Verifying the Installation .............................................. 3-9

3.5 Using the Debugger With the X Window System ..................... 3-10
Using the special keys on the keyboard ......................................... 3-10
Changing the debugger font .......................................................... 3-11
Color mappings on monochrome screens ...................................... 3-11

4 Release Notes ............................................................................ 4-1
Details the features added or changed for this release.
COFF version 2 .............................................................................. 4-1
Multiple MEMORY windows .......................................................... 4-1
Multiple WATCH windows ............................................................. 4-2
New and updated debugger commands ........................................ 4-3
Changes to the TMS320C5xx C Source Debugger User’s Guide ....... 4-4

5 Defining a Memory Map .............................................................. 5-1
Provides instructions for defining and using a memory map to simulate 'C54x ports. The memory map tells the debugger which areas of memory it can and cannot access. This chapter replaces Chapter 6, Defining a Memory Map, in the TMS320C5xx C Source Debugger User’s Guide.

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 ......................................... 5-5
Notes on using the MA command ................................................ 5-6
Memory mapping with the simulator (PCs only) ............................. 5-8

5.4 Customizing the Memory Map .................................................. 5-9
Mapping on-chip dual-access RAM from data memory to program memory .................................................. 5-10
Simulating data memory (ROM) .................................................... 5-10
Programming your memory ......................................................... 5-11
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5 Enabling Memory Mapping</td>
<td>5-12</td>
</tr>
<tr>
<td>5.6 Checking the Memory Map</td>
<td>5-13</td>
</tr>
<tr>
<td>5.7 Modifying the Memory Map During a Debugging Session</td>
<td>5-14</td>
</tr>
<tr>
<td>Returning to the original memory map</td>
<td>5-15</td>
</tr>
<tr>
<td>5.8 Using Multiple Memory Maps for Multiple Target Systems (Emulator Only)</td>
<td>5-16</td>
</tr>
<tr>
<td>5.9 Simulating I/O Space (Simulator Only)</td>
<td>5-17</td>
</tr>
<tr>
<td>Connecting an I/O port</td>
<td>5-17</td>
</tr>
<tr>
<td>Disconnecting an I/O port</td>
<td>5-21</td>
</tr>
<tr>
<td>5.10 Simulating External Interrupts (Simulator Only)</td>
<td>5-22</td>
</tr>
<tr>
<td>Setting up your input file</td>
<td>5-22</td>
</tr>
<tr>
<td>Programming the simulator</td>
<td>5-24</td>
</tr>
<tr>
<td>5.11 Simulating Peripherals (Simulator Only)</td>
<td>5-26</td>
</tr>
<tr>
<td>5.12 Simulating Standard Serial Ports (Simulator Only)</td>
<td>5-27</td>
</tr>
<tr>
<td>Setting up your transmit and receive operations</td>
<td>5-28</td>
</tr>
<tr>
<td>Connecting I/O files</td>
<td>5-29</td>
</tr>
<tr>
<td>Programming the simulator</td>
<td>5-30</td>
</tr>
<tr>
<td>5.13 Simulating Buffered Serial Ports (Simulator Only)</td>
<td>5-31</td>
</tr>
<tr>
<td>Setting up your transmit and receive operations</td>
<td>5-32</td>
</tr>
<tr>
<td>Connecting I/O files</td>
<td>5-33</td>
</tr>
<tr>
<td>Programming the simulator</td>
<td>5-33</td>
</tr>
<tr>
<td>5.14 Simulating TDM Serial Ports (Simulator Only)</td>
<td>5-34</td>
</tr>
<tr>
<td>Setting up your transmit and receive operations</td>
<td>5-35</td>
</tr>
<tr>
<td>Connecting I/O files</td>
<td>5-36</td>
</tr>
<tr>
<td>Programming the simulator</td>
<td>5-36</td>
</tr>
</tbody>
</table>
## Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5–1</td>
<td>Sample Initialization Batch File for Use With the TMS320C54x Simulator</td>
<td>5-4</td>
</tr>
<tr>
<td>5–2</td>
<td>Sample Memory Map for the TMS320C54x Using Memory Cache Capabilities</td>
<td>5-8</td>
</tr>
<tr>
<td>5–3</td>
<td>Connecting Input and Output Ports to Input or Output Files</td>
<td>5-19</td>
</tr>
<tr>
<td>5–4</td>
<td>Connecting an Input Port to an Input File</td>
<td>5-20</td>
</tr>
<tr>
<td>5–5</td>
<td>Using the PINC Command to Connect the Input File</td>
<td>5-24</td>
</tr>
</tbody>
</table>
This chapter helps you install the TMS320C54x simulator and the C source debugger on PC systems running Microsoft™ Windows 3.1. After completing the installation, see the TMS320C5xx C Source Debugger User’s Guide for instructions on using the debugger.

With 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 the debugger requires, the display is clipped. When the display is clipped, it cannot 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 when using the 'C54x simulator.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 System Requirements</td>
<td>1-2</td>
</tr>
<tr>
<td>1.2 Step 1: Installing the Simulator and Debugger Software</td>
<td>1-4</td>
</tr>
<tr>
<td>1.3 Step 2: Setting Up the Debugger Environment</td>
<td>1-5</td>
</tr>
<tr>
<td>1.4 Step 3: Verifying the Installation</td>
<td>1-8</td>
</tr>
</tbody>
</table>
1.1 System Requirements

The following checklists detail items that are shipped with the 'C54x C source debugger and simulator and any additional items you need to use these tools.

**Hardware checklist**

- **Host**
  - An IBM™ PC/AT™ or 100% compatible ISA/EISA-based PC™ with a hard-disk system and a 1.2M-byte floppy-disk drive; a 386 or higher is highly recommended

- **Memory**
  - Minimum of 640K bytes and at least 256K bytes of extended memory

- **Monitor**
  - Monochrome or color monitor (color recommended)

- **Optional hardware**
  - A Microsoft-compatible mouse

- **Miscellaneous materials**
  - Blank, formatted disks

The debugger has several options that allow you to change the overall size of the debugger display. To use a larger screen size, you must invoke the debugger with the appropriate option. For more information about options, see the invocation information in the TMS320C5xx C Source Debugger User’s Guide.
Software checklist

- Operating system
  - Windows version 3.1

- Software tools
  - ’C54x assembler and linker
  - Optional: ’C54x 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 ’C54x memory map. If this file is not present when you invoke the debugger, then all memory is invalid at first. 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, see Chapter 5, Defining a Memory Map.

  - sim54x.cmd batch files (sim541.cmd, sim542.cmd, sim543.cmd, sim545.cmd, sim546.cmd, sim548.cmd, and sim545lp.cmd) contain commands that configure a memory map. Each file simulates a different device—’C541, ’C542, ’C543, ’C545, ’C546, ’C548, or ’C545LP.

  - 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 up the debugger 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, see the information about customizing the debugger display in the TMS320C5xx 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 each product disk.

2) On your hard disk or system disk, create a directory named sim54x. This directory will contain the 'C54x software. Type:

   MD C:\sim54x

3) Insert the debugger product disk into drive A. Copy the contents of the disk:

   COPY A:\*.\* C:\sim54x\*.\* /V

   The Windows version of the debugger executable is called sim54xw.exe.
1.3 Step 2: Setting Up the Debugger Environment

To ensure that your debugger works correctly, you must:

- Modify the PATH statement to identify the sim54x directory.
- Define environment variables so that the debugger can find the files it needs.

**Note:**

Not only must you do these things before you invoke the debugger for the first time, you must do them any time you power up or reboot your PC.

You can accomplish these tasks by entering individual DOS commands, but it is simpler to put the commands in a batch file. You can edit your system's autoexec.bat file to accomplish these tasks. In some cases, however, modifying the autoexec.bat may interfere with other applications running on your PC, so you can create a separate batch file that performs these tasks.

Figure 1–1 (a) shows an example of an autoexec.bat file that contains the suggested modifications (highlighted in bold type). Figure 1–1 (b) shows a sample batch file that you could create instead of editing the autoexec.bat file. For the purpose of discussion, assume that this sample file is named initdb.bat. The subsections following the figure explain these modifications.

**Figure 1–1. DOS-Command Setup for the Debugger**

(a) Sample autoexec.bat file to use with the debugger and simulator

```plaintext
DATE
TIME
ECHO OFF
PATH=
C:\DOS;C:\c5xxtool;C:\sim54x
SET D_DIR=C:\sim54x
SET D_SRC=C:\c54xtool
SET D_OPTIONS=-b
CLS
```

(b) Sample batch file, initdb.bat, to use with the debugger and simulator

```plaintext
PATH=C:\sim54x;%PATH%
SET D_DIR=C:\sim54x
SET D_SRC=C:\c54xtool
SET D_OPTIONS=-b
```
Modifying the PATH statement

Define a path to the debugger directory. The general format for doing this is:

\[ \text{PATH}=\text{C:\sim54x} \]

This allows you to invoke the debugger without specifying the name of the directory that contains the debugger executable file.

- If you are modifying an autoexec.bat that already contains a PATH statement, simply include \[ ;\text{C:\sim54x} \] at the end of the statement, as shown in Figure 1–1 (a).

- If you are creating an initdb.bat file, use a different format for the PATH statement:

\[ \text{PATH}=\text{C:\sim54x;%PATH%} \]

The addition of \[ ;\text{%PATH%} \] ensures that this PATH statement will not undo PATH statements in any other batch files (including the autoexec.bat file).

Setting up the environment variables

An environment variable is a special system symbol that a program uses for finding or obtaining certain types of information. The debugger uses three environment variables, named D_DIR, D_SRC, and D_OPTIONS. Set up these environment variables in your batch file as described in the following list. The format for doing this is the same whether you edit the autoexec.bat file or create an initdb.bat file.

- Identify the sim54x directory with D_DIR. Enter:

  \[ \text{SET D_DIR}=\text{C:\sim54x} \]

  (Be careful not to precede the equal sign with a space.)

  This directory contains auxiliary files (such as siminit.cmd) that the debugger needs.

- Identify with D_SRC any directories that contain program source files that you want to look at while you are debugging code. The general format for doing this is:

  \[ \text{SET D_SRC}=\text{pathname}_1;\text{pathname}_2;\ldots \]

  (Be careful not to precede the equal sign with a space.)

  For example, if your 'C54x programs were in a directory named csourse on drive C, the D_SRC setup would be:

  \[ \text{SET D_SRC}=\text{C:\csourse} \]
Identify with D_OPTIONS the invocation options that you want to use regularly. Use this format:

```
SET D_OPTIONS= [filename] [options]
```

(Be careful not to precede the equal sign with a space.)

The filename identifies the optional object file for the debugger to load, and options list the options you want to use at invocation. These are the options that you can identify with D_OPTIONS:

<table>
<thead>
<tr>
<th>Option</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>–b</td>
<td>Select a screen size of 80 characters by 43 lines (EGA or VGA)</td>
</tr>
<tr>
<td>–bb</td>
<td>Select a screen size of 80 characters by 50 lines (VGA only)</td>
</tr>
<tr>
<td>–bl#</td>
<td>Select a screen length of # lines (default is 25)</td>
</tr>
<tr>
<td>–bw#</td>
<td>Select a screen width of # characters (default is 80)</td>
</tr>
<tr>
<td>–i pathname</td>
<td>Identify additional directories</td>
</tr>
<tr>
<td>–min</td>
<td>Select the minimal debugging mode</td>
</tr>
<tr>
<td>–mv version</td>
<td>Specify the memory map to use with the simulator</td>
</tr>
<tr>
<td>–profile</td>
<td>Enter profiling environment</td>
</tr>
<tr>
<td>–s</td>
<td>Load the symbol table only</td>
</tr>
<tr>
<td>–t filename</td>
<td>Identify a new initialization file</td>
</tr>
<tr>
<td>–v</td>
<td>Load without the symbol table</td>
</tr>
</tbody>
</table>

You can override D_OPTIONS by invoking the debugger with the –x option.

For more information about options, see the invocation instructions in the TMS320C5xx C Source Debugger User’s Guide.

**Invoking the modified or new batch file**

- If you modify the autoexec.bat file, be sure to invoke it before invoking the debugger for the first time. To invoke this file, enter:

  `AUTOEXEC`

- If you create an initdb.bat file, you must invoke it before entering Windows. You must invoke initdb.bat any time that you power up or reboot your PC. To invoke this file, enter:

  `INITDB`
1.4 Step 3: Verifying the Installation

To ensure that you have correctly installed the simulator and debugger software, follow these steps:

1) Start Windows.

2) In the Program Manager or File Manager, select Run... from the File menu.

3) In the Command Line field of the Run dialog box, enter:
   \[ c:\sim54x\sim54xw \text{sample} \]

You should see a display similar to this one:

If you do not see a 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.
This chapter helps you install the TMS320C54x simulator and the C source debugger on a SPARCstation running SunOS. After completing the installation, see the TMS320C5xx C Source Debugger User’s Guide for instructions on using the debugger.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 System Requirements</td>
<td>2-2</td>
</tr>
<tr>
<td>2.2 Step 1: Installing the Simulator and Debugger Software</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3 Step 2: Setting Up the Debugger Environment</td>
<td>2-6</td>
</tr>
<tr>
<td>2.4 Step 3: Verifying the Installation</td>
<td>2-9</td>
</tr>
<tr>
<td>2.5 Using the Debugger With the X Window System</td>
<td>2-10</td>
</tr>
</tbody>
</table>
2.1 System Requirements

The following checklists detail items that are shipped with the 'C54x C source debugger and simulator and additional items you need to use these tools.

**Hardware checklist**

- **Host**: A SPARCstation or a system that is 100% compatible with a SPARCstation 2 class or higher
- **Monitor**: Monochrome or color monitor (color recommended)
- **Disk space**: 2M bytes of disk space
- **Required hardware**: CD-ROM drive
- **Optional hardware**: Mouse

**Software checklist**

- **Operating system**: SunOS version 4.1.3 (or higher) or SunOS version 5.x (also known as Solaris™ 2.x) using an X Window System type window manager, such as OpenWindows™ version 3.0 (or higher). If you are using SunOS 5.x, you must have the Binary Compatibility Package (BCP) installed; if you don’t, get your system administrator’s help.
- **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 do not have root privileges, get help from your system administrator.
- **Software tools**: 'C54x assembler and linker
  Optional: 'C54x 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 'C54x memory map. If this file is not present when you invoke the debugger, then all memory is invalid at first. 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, see Chapter 5, *Defining a Memory Map*.
  - `sim54x.cmd` batch files (`sim541.cmd`, `sim542.cmd`, `sim543.cmd`, `sim545.cmd`, `sim546.cmd`, and `sim548.cmd`, and `sim545lp.cmd`) contain commands that configure a memory map. Each file simulates a different device—’C541, ’C542, ’C543, ’C545, ’C546, ’C548, or ’C545LP.
  - `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 up the debugger 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, see the information about customizing the debugger display in the TMS320C5xx 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.

**Note:**
If you are running SunOS 4.1.x, 5.0, or 5.1, you must have root privileges to mount or unmount the CD-ROM. If you do not have root privileges, get help from your system administrator.

**Mounting the CD-ROM**

The steps to mount the CD-ROM vary according to your operating system version:

- If you have a SunOS 4.1.x, load the CD-ROM into the drive. As root, 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. As root, 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
    ```
Step 1: Installing the Simulator and Debugger Software

Copying the files

After you have mounted 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 sim54x on your hard disk. To create this directory, enter:
   
   \texttt{mkdir /your\_pathname/sim54x}

2) Copy the files from the CD-ROM to your hard-disk system:
   
   \texttt{cp -r \* /your\_pathname/sim54x}

Unmounting the CD-ROM

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

- If you have SunOS 4.1.x, 5.0, or 5.1, as root, enter the following from a command shell:
  
  \texttt{cd}
  \texttt{umount /cdrom}
  \texttt{eject /dev/sr0}
  \texttt{exit}

- If you have SunOS 5.2 or higher, enter the following from a command shell:
  
  \texttt{cd}
  \texttt{eject}
2.3 Step 2: Setting Up the Debugger Environment

To ensure that your debugger works correctly, you must:

- Modify the shell path variable to include the sim54x directory.
- Define environment variables so that the debugger can find the files it needs.
- Reinitialize your shell.

Modifying the path statement

You must include the debugger directory in your shell path. To do this, you must modify the shell configuration file in your home directory (for example, the .cshrc file for a C shell). This file must include the pathname to your sim54x directory in your path if it is not already there. The following statement is an example of what a typical path-variable definition looks like:

```bash
set path = (. /bin /usr/ucb /usr/contrib/bin /usr/bin \\
/usr/openwin/bin)
```

Following is an example of that path variable modified to include the pathname to sim54x. The part of the path in bold type is the modification:

```bash
set path = (. /bin /usr/ucb /usr/contrib/bin /usr/bin \\
/usr/openwin/bin /user/fred/sim54x)
```

You would use the path to your home directory in place of /user/fred.

Setting up the environment variables

An environment variable is a special system symbol that a program uses for finding or obtaining certain types of information. The debugger uses four environment variables, named D_DIR, D_SRC, D_OPTIONS, and DISPLAY (X Window System™ only). You can set up these environment variables in your shell configuration file. Follow these steps to set up the environment variables:

- Identify the sim54x directory with D_DIR. This directory contains auxiliary files (such as siminit.cmd) that the debugger needs. The general format for doing this is:
  ```bash
  setenv D_DIR "pathname"
  ```
  For example, if the files are in a directory named /user/fred/sim54x, the D_DIR setup would be:
  ```bash
  setenv D_DIR "/user/fred/sim54x"
  ```
  (Be sure to enclose the directory name within quotes.)
Identify with D_SRC any directories that contain program source files that you want to look at while you are debugging code. The general format for doing this is:

```
setenv D_SRC "pathname1;pathname2..."
```

(Be sure to enclose the path names within one set of quotes.)

For example, if your 'C54x programs were in a directory named /user/fred/c54xsource, the D_SRC setup would be:

```
setenv D_SRC "/user/fred/c54xsource"
```

Identify with D_OPTIONS the invocation options that you want to use regularly. Use this format:

```
setenv D_OPTIONS ["filename [options]"
```

(Be sure to enclose the filename and options within one set of quotes.)

The filename identifies the optional object file for the debugger to load, and options list the options you want to use at invocation. These are the options that you can identify with D_OPTIONS:

<table>
<thead>
<tr>
<th>Option</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>–b</td>
<td>Select a screen size of 80 characters by 43 lines (EGA or VGA)</td>
</tr>
<tr>
<td>–bb</td>
<td>Select a screen size of 80 characters by 50 lines (VGA only)</td>
</tr>
<tr>
<td>–d machine name</td>
<td>Display debugger on a different machine</td>
</tr>
<tr>
<td>–i pathname</td>
<td>Identify additional directories</td>
</tr>
<tr>
<td>–min</td>
<td>Select the minimal debugging mode</td>
</tr>
<tr>
<td>–mv version</td>
<td>Specify the memory map to use with the simulator</td>
</tr>
<tr>
<td>–profile</td>
<td>Enter profiling environment</td>
</tr>
<tr>
<td>–s</td>
<td>Load the symbol table only</td>
</tr>
<tr>
<td>–t filename</td>
<td>Identify a new initialization file</td>
</tr>
<tr>
<td>–v</td>
<td>Load without the symbol table</td>
</tr>
</tbody>
</table>

You can override D_OPTIONS by invoking the debugger with the --x option.

For more information about options, see the invocation instructions in the TMS320C5xx C Source Debugger User's Guide.
Step 2: Setting Up the Debugger Environment

- If you are using the X Window System, you can display the debugger on a different machine than the one the parallel debug manager and simulator core are running on. To do so, you need to set up two environment variables:
  - Be sure that the LD_LIBRARY_PATH environment variable is set to the following:
    ```
    LD_LIBRARY_PATH $OPENWINHOME/lib
    ```
    If the LD_LIBRARY_PATH variable is not set correctly, use this command:
    ```
    setenv LD_LIBRARY_PATH "$OPENWINHOME/lib"
    ```
  - Set up the DISPLAY environment variable. The general format for doing this is:
    ```
    setenv DISPLAY "machinename"
    ```
    You can also specify a different machine by using the –d debugger option (see the TMS320C5xx C Source Debugger User’s Guide for more information). If you use both the DISPLAY environment variable and –d, the –d option overrides DISPLAY.

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:

```bash
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:

```
sim54x sample
```

You should see a display similar to this one:

If you do not see a 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 use the X Window System to run the `C54x debugger, you need to know about the keyboard's special keys, the debugger font, and using the debugger on a monochrome monitor.

**Using the special keys on the keyboard**

The debugger uses some special keys that you can map differently than 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, but the functions are available.

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.

<table>
<thead>
<tr>
<th>Debugger Key Needed</th>
<th>Keysym for That Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1 to F10</td>
<td>F1 to F10</td>
</tr>
<tr>
<td>PAGE UP</td>
<td>Prior</td>
</tr>
<tr>
<td>PAGE DOWN</td>
<td>Next</td>
</tr>
<tr>
<td>HOME</td>
<td>Home</td>
</tr>
<tr>
<td>END</td>
<td>End</td>
</tr>
<tr>
<td>INSERT</td>
<td>Insert</td>
</tr>
<tr>
<td>→</td>
<td>Right</td>
</tr>
<tr>
<td>←</td>
<td>Left</td>
</tr>
<tr>
<td>↑</td>
<td>Up</td>
</tr>
<tr>
<td>↓</td>
<td>Down</td>
</tr>
</tbody>
</table>

Use the X utility xev to check the keysyms 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 map a Sun Type 4 keyboard to the keys listed above:

```bash
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 'C54x debugger font to Courier, add the following line to the .Xdefaults file:

```
sim54x*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, you can use a monochrome monitor. The following table shows the color mappings for monochrome screens:

<table>
<thead>
<tr>
<th>Color</th>
<th>Appearance on Monochrome Screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>black</td>
<td>black</td>
</tr>
<tr>
<td>blue</td>
<td>black</td>
</tr>
<tr>
<td>green</td>
<td>white</td>
</tr>
<tr>
<td>cyan</td>
<td>white</td>
</tr>
<tr>
<td>red</td>
<td>black</td>
</tr>
<tr>
<td>magenta</td>
<td>black</td>
</tr>
<tr>
<td>yellow</td>
<td>white</td>
</tr>
<tr>
<td>white</td>
<td>white</td>
</tr>
</tbody>
</table>
Installing the Simulator and C Source Debugger With HP-UX

This chapter helps you install the TMS320C54x simulator and the C source debugger on a HP 9000 series 700™ PA-RISC™ system running HP-UX. After completing the installation, see the TMS320C5xx C Source Debugger User's Guide for instructions on using the debugger.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 System Requirements</td>
<td>3-2</td>
</tr>
<tr>
<td>3.2 Step 1: Installing the Simulator and Debugger Software</td>
<td>3-4</td>
</tr>
<tr>
<td>3.3 Step 2: Setting Up the Debugger Environment</td>
<td>3-6</td>
</tr>
<tr>
<td>3.4 Step 3: Verifying the Installation</td>
<td>3-9</td>
</tr>
<tr>
<td>3.5 Using the Debugger With the X Window System</td>
<td>3-10</td>
</tr>
</tbody>
</table>
3.1 System Requirements

The following checklists detail items that are shipped with the 'C54x C source
debugger and simulator and additional items you need to use these tools.

**Hardware checklist**

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>An HP 9000 Series 700 PA-RISC system</td>
</tr>
<tr>
<td>Monitor</td>
<td>Monochrome or color (color recommended)</td>
</tr>
<tr>
<td>Disk space</td>
<td>2M bytes of disk space</td>
</tr>
<tr>
<td>Required hardware</td>
<td>CD-ROM drive</td>
</tr>
<tr>
<td>Optional hardware</td>
<td>Mouse</td>
</tr>
</tbody>
</table>
System Requirements

Software checklist

- Operating system: HP-UX 9.x or later
- Root privileges: You must have root privileges to mount and unmount the CD-ROM. If you do not have root privileges, get help from your system administrator.
- Software tools: ‘C54x assembler and linker
  Optional: ‘C54x 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 ‘C54x memory map. If this file is not present when you invoke the debugger, then all memory is invalid at first. 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, see Chapter 5, Defining a Memory Map.
  - sim54x.cmd batch files (sim541.cmd, sim542.cmd, sim543.cmd, sim545.cmd, sim546.cmd, sim548.cmd, and sim545lp.cmd) contain commands that configure a memory map. Each file simulates a different device—‘C541, ‘C542, ‘C543, ‘C545, ‘C546, ‘C548, or ‘C545LP.
  - 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 up the debugger 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, see the information about customizing the debugger display in the TMS320C5xx 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.

Note:
You must have root privileges to mount or unmount the CD-ROM. If you do not have root privileges, get help from your system administrator.

Mounting the CD-ROM

As root, you can mount the CD-ROM using the UNIX™ mount command or the SAM (system administration manager):

- To use the UNIX mount command, enter:
  ```
  mount -rt cdfs /dev/dsk/your_cdrom_device /cdrom
  exit
  ```
  Make the hp directory on the CD-ROM the current directory. For example, if the CD-ROM is mounted at /cdrom, enter:
  ```
  cd /cdrom/hp
  ```

- To use SAM to mount the CD-ROM, see the instructions in the HP documentation about SAM.

Copying the files and setting up the simulator

After you have mounted 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 sim54x on your hard disk. To create this directory, enter:
   ```
   mkdir sim54x
   ```

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

3) Copy the files from the CD-ROM to your hard-disk system:
   ```
   cp -r * sim54x
   ```
Step 1: Installing the Simulator and Debugger Software

**Unmounting the CD-ROM**

You must unmount the CD-ROM after copying the files. As root, enter:

```
  cd
  umount /cdrom
  exit
```
Step 2: Setting Up the Debugger Environment

3.3 Step 2: Setting Up the Debugger Environment

To ensure that your debugger works correctly, you must:

- Modify the shell path variable to include the sim54x directory.
- Define environment variables so that the debugger can find the files it needs.
- Reinitialize your shell.

Modifying the path statement

You must include the debugger directory in your shell path. To do this, you must modify the shell configuration file in your home directory (for example, the .cshrc file for a C shell). This file must include the pathname to your sim54x directory in your path if it is not already there. The following statement is an example of what a typical path-variable definition looks like:

```
set path = (. /bin /usr/ucb /usr/contrib/bin /usr/bin \\
/usr/openwin/bin)
```

Following is an example of that path variable modified to include the pathname to sim54x. The part of the path in bold type is the modification:

```
set path = (. /bin /usr/ucb /usr/contrib/bin /usr/bin \\
/usr/openwin/bin /user/fred/sim54x)
```

You would use the path to your home directory in place of /user/fred.

Setting up the environment variables

An environment variable is a special system symbol that the debugger uses for finding or obtaining certain types of information. The debugger uses four environment variables, named D_DIR, D_SRC, D_OPTIONS, and DISPLAY (X Window System only). You can set up these environment variables in your shell configuration file. Follow these steps to set up the environment variables:

- Identify the sim54x directory with D_DIR. This directory contains auxiliary files (such as siminit.cmd) that the debugger needs. The general format for doing this is:

  ```
  setenv D_DIR "pathname"
  ```

  For example, if the files are in a directory named /user/fred/sim54x, the D_DIR setup would be:

  ```
  setenv D_DIR "/user/fred/sim54x"
  ```

  (Be sure to enclose the directory name within quotes.)
Step 2: Setting Up the Debugger Environment

- Identify any directories that contain program source files that you want to look at while you are debugging code with D_SRC. The general format for doing this is:

  ```bash
  setenv D_SRC "pathname1;pathname2;...
  ```

  (Be sure to enclose the path names within one set of quotes.)

  For example, if your C54x programs were in a directory named /user/fred/c54xsource, the D_SRC setup would be:

  ```bash
  setenv D_SRC "/user/fred/c54xsource"
  ```

- Identify with D_OPTIONS the invocation options that you want to use regularly. Use this format:

  ```bash
  setenv D_OPTIONS "[filename] [options]"
  ```

  (Be sure to enclose the filename and options within one set of quotes.)

  The filename identifies the optional object file for the debugger to load, and options list the options you want to use at invocation. These are the options that you can identify with D_OPTIONS:

<table>
<thead>
<tr>
<th>Option</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>–b</td>
<td>Select a screen size of 80 characters by 43 lines</td>
</tr>
<tr>
<td>–bb</td>
<td>(EGA or VGA)</td>
</tr>
<tr>
<td>–d machine</td>
<td>Display debugger on a different machine</td>
</tr>
<tr>
<td>–i pathname</td>
<td>Identify additional directories</td>
</tr>
<tr>
<td>–min</td>
<td>Select the minimal debugging mode</td>
</tr>
<tr>
<td>–mv version</td>
<td>Specify the memory map to use with the simulator</td>
</tr>
<tr>
<td>–profile</td>
<td>Enter profiling environment</td>
</tr>
<tr>
<td>–s</td>
<td>Load the symbol table only</td>
</tr>
<tr>
<td>–t filename</td>
<td>Identify a new initialization file</td>
</tr>
<tr>
<td>–v</td>
<td>Load without the symbol table</td>
</tr>
</tbody>
</table>

  You can override D_OPTIONS by invoking the debugger with the –x option.

  For more information about options, see the invocation instructions in the TMS320C5xx C Source Debugger User’s Guide.
Step 2: Setting Up the Debugger Environment

If you are using the X Window System, you can display the debugger on a different machine than the one the parallel debug manager and simulator core are running on. To do so, you need to set up two environment variables:

- Be sure that the LD_LIBRARY_PATH environment variable is set to the following:
  
  ```bash
  LD_LIBRARY_PATH $OPENWINHOME/lib
  ```

  If the LD_LIBRARY_PATH variable is not set correctly, use this command:

  ```bash
  setenv LD_LIBRARY_PATH "$OPENWINHOME/lib"
  ```

- Set up the DISPLAY environment variable. The general format for doing this is:

  ```bash
  setenv DISPLAY "machinename"
  ```

  You can also specify a different machine by using the –d debugger option (see the TMS320C5xx C Source Debugger User’s Guide for more information). If you use both the DISPLAY environment variable and –d, the –d option overrides DISPLAY.

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:

```bash
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:

\[ \text{sim54x sample} \]

You should see a display similar to this one:

If you do not see a 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 use the X Window System to run the 'C54x debugger, you need to know about the keyboard’s special keys, the debugger font, and using the debugger on a monochrome monitor.

Using the special keys on the keyboard

The debugger uses some special keys that you can map differently than 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, but the functions are available.

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.

<table>
<thead>
<tr>
<th>Debugger Key Needed</th>
<th>Keysym for That Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1 to F10</td>
<td>F1 to F10</td>
</tr>
<tr>
<td>PAGE UP</td>
<td>Prior</td>
</tr>
<tr>
<td>PAGE DOWN</td>
<td>Next</td>
</tr>
<tr>
<td>HOME</td>
<td>Home</td>
</tr>
<tr>
<td>END</td>
<td>End</td>
</tr>
<tr>
<td>INSERT</td>
<td>Insert</td>
</tr>
<tr>
<td>→</td>
<td>Right</td>
</tr>
<tr>
<td>←</td>
<td>Left</td>
</tr>
<tr>
<td>↑</td>
<td>Up</td>
</tr>
<tr>
<td>↓</td>
<td>Down</td>
</tr>
</tbody>
</table>

Use the X utility xev to check the keysyms 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 map a Sun Type 4 keyboard to the keys listed above:

```sh
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 'C54x debugger font to Courier, add the following line to the `.Xdefaults` file:

```
sim54x*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, you can use a monochrome monitor. The following table shows the color mappings for monochrome screens:

<table>
<thead>
<tr>
<th>Color</th>
<th>Appearance on Monochrome Screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>black</td>
<td>black</td>
</tr>
<tr>
<td>blue</td>
<td>black</td>
</tr>
<tr>
<td>green</td>
<td>white</td>
</tr>
<tr>
<td>cyan</td>
<td>white</td>
</tr>
<tr>
<td>red</td>
<td>black</td>
</tr>
<tr>
<td>magenta</td>
<td>black</td>
</tr>
<tr>
<td>yellow</td>
<td>white</td>
</tr>
<tr>
<td>white</td>
<td>white</td>
</tr>
</tbody>
</table>
This release of the TMS320C54x debugger contains general enhancements as well as enhancements specific to the 'C54x simulator version of the debugger. The following sections describe these enhancements.

**COFF version 2**

This release supports an expanded object file format called COFF2. The debugger can handle COFF object files developed with assembly language tools using the COFF0, COFF1, or COFF2 formats.

**Multiple MEMORY windows**

You can now open as many MEMORY windows as you want. The MEM command has a new, optional window name parameter. When you open an additional MEMORY window using the window name parameter, the debugger appends the window name to the MEMORY window label. The new basic syntax for the MEM command is:

\[ \text{mem } \text{expression [, display format] [, window name]} \]

You can use the MEM command to:

- Open an additional MEMORY window
- Display a new memory range in an open MEMORY window

The window name parameter is optional if you are displaying a different memory range in the default MEMORY window. Use the window name parameter when you want to display a new memory range in one of the additional MEMORY windows.
Multiple WATCH windows

You can now access multiple WATCH windows. Use the window name parameter as described for each WATCH window command.

- The WA command has a new, optional window name parameter. When you open a WATCH window using the window name parameter, the debugger appends the window name to the WATCH window label. You can create as many WATCH windows as you need. The basic syntax for the WA command is:
  wa expression [, label] [, display format] [, , window name]

  If you omit the window name parameter, the debugger displays the expression in the default WATCH window (labeled WATCH).

- The WD command deletes a specific item from the WATCH window. The WD command’s index number parameter must correspond to one of the watch indexes listed in the WATCH window. The optional window name parameter is used to specify a particular WATCH window. If you do not use the window name parameter, the WD command deletes the item from the default WATCH window. The basic syntax for the WD command is:
  wd index number [, , window name]

- The WR command deletes all items from a WATCH window and closes the window.
  - To close the default WATCH window, enter:
    wr
  - To close one of the additional WATCH windows, use this syntax:
    wr window name
  - To close all WATCH windows, enter:
    wr *
New and updated debugger commands

The debugger now supports the following commands on all platforms.

**cd, chdir**  
Change Directory

**Syntax**  
`cd` [directory name]  
`chdir` [directory name]

**Menu selection**  
none

**Environments**  
✓ basic debugger  
☐ PDM  
✓ profiling

**Description**  
The CD or CHDIR command changes the current working directory from within the debugger. You can use relative pathnames as part of the directory name. If you don’t use a directory name, the CD command displays the name of the current directory. This command can affect any other command whose parameter is a filename, such as the FILE, LOAD, and TAKE commands, when used with the USE command. You can also use the CD command to change the current drive. For example:

```
  cd c:  
cd d:\csource  
cd c:\sim54x
```

**dir**  
List Directory Contents

**Syntax**  
`dir` [directory name]

**Menu selection**  
none

**Environments**  
✓ basic debugger  
☐ PDM  
✓ profiling

**Description**  
The DIR command displays a directory listing in the display area of the COMMAND window. If you use the optional directory name parameter, the debugger displays a list of the specified directory’s contents. If you don’t use a directory name, the debugger lists the contents of the current directory.

You can list only files that match a specific format within a directory by using the asterisk (*) wildcard character. If the directory name ends in a partial filename with an asterisk, the debugger lists only the files which match the wildcard string. For example, to list every file in the home directory that has a .cmd extension, you would enter:

```
  DIR /home/* .cmd
```
safehalt

Safehalt Mode

Syntax

safehalt {on | off}

Menu selection

none

Environments

basic debugger
PDM
profiling

Description

This new command, SAFEHALT, places the debugger in safehalt mode. When safehalt mode is off (the default), you can halt a running target device either by pressing \texttt{ESC} or by clicking a mouse button. When safehalt mode is on, you can halt a running target device only by pressing \texttt{ESC}; mouse clicks are ignored.

Changes to the TMS320C5xx C Source Debugger User's Guide

The Debugger Options section in the TMS320C5xx C source Debugger User’s Guide describes the options that you can use when invoking the debugger. The –mv option has been added for the simulator version of the debugger.

The –mv option specifies which memory map the simulator loads. By default, the simulator loads the memory map contained in the siminit.cmd file, which is a generic memory map. Each of the provided memory maps simulates a different ’C54x device, as described in the following table:

<table>
<thead>
<tr>
<th>Option</th>
<th>Device Simulated</th>
<th>Initialization File Used</th>
<th>Peripherals Simulated</th>
</tr>
</thead>
<tbody>
<tr>
<td>–mv541</td>
<td>’C541</td>
<td>sim541.cmd</td>
<td>Serial port 0, serial port 1, timer</td>
</tr>
<tr>
<td>–mv542</td>
<td>’C542</td>
<td>sim542.cmd</td>
<td>Buffered serial port, TDM serial port, timer</td>
</tr>
<tr>
<td>–mv543</td>
<td>’C543</td>
<td>sim543.cmd</td>
<td>Buffered serial port, TDM serial port, timer</td>
</tr>
<tr>
<td>–mv545</td>
<td>’C545</td>
<td>sim545.cmd</td>
<td>Buffered serial port, serial port 1, timer</td>
</tr>
<tr>
<td>–mv546</td>
<td>’C546</td>
<td>sim546.cmd</td>
<td>Buffered serial port, serial port 1, timer</td>
</tr>
<tr>
<td>–mv548</td>
<td>’C548</td>
<td>sim548.cmd</td>
<td>2 Buffered serial ports, TDM serial port, timer, HPI</td>
</tr>
<tr>
<td>–mv545lp</td>
<td>’C545LP</td>
<td>sim545lp.cmd</td>
<td>Buffered serial port, serial port 1, timer, HPI</td>
</tr>
</tbody>
</table>
Defining a Memory Map

Note:
This chapter replaces Chapter 6, Defining a Memory Map, in the TMS320C5xx C Source Debugger User’s Guide.

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 cannot access. You can use the Memory pulldown menu to enter the commands described in this chapter.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 The Memory Map: What It Is and Why You Must Define It</td>
<td>5-2</td>
</tr>
<tr>
<td>5.2 A Sample Memory Map</td>
<td>5-4</td>
</tr>
<tr>
<td>5.3 Identifying Usable Memory Ranges</td>
<td>5-5</td>
</tr>
<tr>
<td>5.4 Customizing the Memory Map</td>
<td>5-9</td>
</tr>
<tr>
<td>5.5 Enabling Memory Mapping</td>
<td>5-12</td>
</tr>
<tr>
<td>5.6 Checking the Memory Map</td>
<td>5-13</td>
</tr>
<tr>
<td>5.7 Modifying the Memory Map During a Debugging Session</td>
<td>5-14</td>
</tr>
<tr>
<td>5.8 Using Multiple Memory Maps for Multiple Target Systems (Emulator Only)</td>
<td>5-16</td>
</tr>
<tr>
<td>5.9 Simulating I/O Space (Simulator Only)</td>
<td>5-17</td>
</tr>
<tr>
<td>5.10 Simulating External Interrupts (Simulator Only)</td>
<td>5-22</td>
</tr>
<tr>
<td>5.11 Simulating Peripherals (Simulator Only)</td>
<td>5-26</td>
</tr>
<tr>
<td>5.12 Simulating Standard Serial Ports (Simulator Only)</td>
<td>5-27</td>
</tr>
<tr>
<td>5.13 Simulating Buffered Serial Ports (Simulator Only)</td>
<td>5-31</td>
</tr>
<tr>
<td>5.14 Simulating TDM Serial Ports (Simulator Only)</td>
<td>5-34</td>
</tr>
</tbody>
</table>
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 cannot 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 cannot 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 are using the debugger. However, this can be inconvenient because, in most cases, you will set up one memory map before you begin debugging and will use this map for all of your debugging sessions. The easiest method of 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 the 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 whether you have 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 have not used the –t option, the debugger looks for the default initialization batch file. The batch filename differs for each version of the debugger:

- For the emulator, this file is called emuinit.cmd.
- For the EVM, this file is called evminit.cmd.
- For the simulator, this file is called siminit.cmd.

If the debugger finds the file corresponding to your tool, it executes the file.

3) If the debugger does not find the –t option or the initialization batch file, it looks for a file called init.cmd. This search mechanism allows you to have one initialization batch file for more than one debugger tool. To set up this file, you can use the IF/ELSE/ENDIF commands (for more details, see the Entering and Using Commands chapter in the TMS320C5xx C Source Debugger User’s Guide) to indicate which memory map applies to each tool.

**Potential memory map problems**

You may experience these problems if the memory map is not correctly defined and enabled:

- **Accessing invalid memory addresses.** If you do not 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 the Debugger and PDM Messages appendix in the TMS320C5xx C Source Debugger User’s Guide.

- **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 5-5). Alternatively, you can turn memory mapping off during a load by using the MAP OFF command (see page 5-12).

- **Accessing conflict and extra cycles (simulator only).** If two memory read access requests occur simultaneously during an execution, the simulator may be unable to complete both requests within the same clock cycle. If both locations belong to the same physical memory block and the block is single-access memory, both requests cannot be processed within the same clock cycle.
5.2 A Sample Memory Map

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

Example 5–1. Sample Initialization Batch File for Use With the TMS320C54x Simulator

```
ma 0x0000, 0, 0x80, EX|RAM
ma 0xc000, 0, 0x1000, ROM
ma 0xd000, 0, 0x1000, EX|RAM

ma 0x0000, 1, 0x0060, RAM
ma 0x0060, 1, 0x0020, RAM
ma 0x0080, 1, 0x0380, RAM|DA
ma 0x0400, 1, 0x0400, EX|RAM
```

The MA commands (shown in Example 5–1) define valid memory ranges and identify the read/write characteristics of the memory ranges. The MAP command enables mapping (see Section 5.5, Enabling Memory Mapping, on page 5-12). By default, mapping is enabled when you invoke the debugger. Figure 5–1 illustrates the memory map defined in Example 5–1.

Figure 5–1. Sample Memory Map for Use With the TMS320C54x Simulator

<table>
<thead>
<tr>
<th>Program memory</th>
<th>Data memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 to 0x007F</td>
<td>0x0000 to 0x005F</td>
</tr>
<tr>
<td>0x0080 to 0xBFFF</td>
<td>0x0060 to 0x007F</td>
</tr>
<tr>
<td>0xC000 to 0xCFFF</td>
<td>0x0080 to 0x03FF</td>
</tr>
<tr>
<td>0xD000 to 0xDFFF</td>
<td>0x0400 to 0x07FF</td>
</tr>
<tr>
<td>0xE000 to 0xFFFF</td>
<td>0x0800 to 0xFFFF</td>
</tr>
</tbody>
</table>
5.3 Identifying Usable Memory Ranges

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

```
ma address, page, length, type
```

- The `address` parameter defines the starting address of a range. This parameter can be 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 display area of the COMMAND window:

```
Conflicting map range
```

- The `page` parameter is a 1-digit number that identifies the type of memory (program, data, or I/O) that a range occupies:

<table>
<thead>
<tr>
<th>To identify this page . . .</th>
<th>Use this value as the page parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program memory</td>
<td>0</td>
</tr>
<tr>
<td>Data memory</td>
<td>1</td>
</tr>
<tr>
<td>I/O space</td>
<td>2</td>
</tr>
</tbody>
</table>

- The `length` parameter defines the length of the range. This parameter can be any C expression.

- The `type` parameter identifies the read/write characteristics of the memory range. The `type` must be one of these keywords:

<table>
<thead>
<tr>
<th>To identify this kind of memory . . .</th>
<th>Use this keyword as the type parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-only memory</td>
<td>R or ROM</td>
</tr>
<tr>
<td>Write-only memory</td>
<td>W or WOM</td>
</tr>
<tr>
<td>Read/write memory</td>
<td>R</td>
</tr>
<tr>
<td>Read/write external memory</td>
<td>RAM</td>
</tr>
<tr>
<td>Read-only port</td>
<td>P</td>
</tr>
<tr>
<td>Read/write port</td>
<td>P</td>
</tr>
<tr>
<td>Single-access memory</td>
<td>SA</td>
</tr>
<tr>
<td>Dual-access memory</td>
<td>DA</td>
</tr>
</tbody>
</table>
Notes on using the MA command

- The debugger caches memory that is not defined as a port type (P|R, P|W, or P|R|W). For ranges that you do not want cached, be sure to map them as ports.

- When you are using the simulator, you can use the parameter values P|R, P|W, and P|R|W to simulate I/O ports. See Section 5.9, Simulating I/O Space, on page 5-17.

- Be sure that the map ranges that you specify in a common object file format (COFF) file match those that you define with the MA command. Moreover, a command sequence such as:

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

  does not equal

  ma x,y+z,ram

  If you were planning to load two COFF blocks, where the first block spanned the length of y and the second block spanned the length of z, you would use the first MA command example. However, if you were planning to load a COFF block that spanned the length of y + z, you would use the second MA command example.

  Alternatively, you could turn memory mapping off during a load by using the MAP OFF command. Although the MAP OFF command can be useful, you need to be sure that you use it correctly. See Section 5.5, Enabling Memory Mapping, on page 5-12 for more information about using the MAP OFF command.

- Although the address range for both of the following MA commands is the same (0x0400 to 0x0800), one range is internal and the other range is external.

  ma 0x0400, 0, 0x0800, ROM
  ma 0x0400, 0, 0x0800, EX|ROM

  When the simulator is operating in microcomputer mode (MP/MC = 0), the internal program ROM is accessed. Otherwise, the external program memory module is used.

- If a range of memory is configured as dual-access RAM (using the DA attribute with the MA command), it means two simultaneous accesses (read/write) can be performed during the same cycle to the block.

  For example, the following command creates one dual-access RAM as a data page. If an instruction performs two simultaneous accesses to two addresses in this block, both accesses execute in one cycle.

  ma 0x0100, 1, 0x0100, R|W|DA
If a range of memory is configured as single-access RAM (using the SA attribute with the MA command), it means only one access (read/write) can be performed on any address in the block in one cycle. You can configure more than one single-access RAM block. Simultaneous accesses to different single-access RAM blocks during the same cycle are permitted.

For example, the following commands create two single-access RAM blocks. The blocks are 0x100 in size. If an instruction performs two accesses, one in the first block (for example, address 0x110) and another in the second block (for example, address 0x230), the instruction executes in only one cycle.

```
ma 0x0100, 1, 0x0100, R|W|SA
ma 0x0200, 1, 0x0100, R|W|SA
```

Contrarily, if the blocks were combined into one block and configured as one single block of 0x200 words (as shown in the following command), simultaneous accesses to addresses 0x110 and 0x230 would take two cycles to complete.

```
ma 0x100, 1, 0x200, R|W|SA
```
Memory mapping with the simulator (PCs only)

Unlike the emulator and EVM, the 'C54x 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 'C54x, do not allocate more than 20K words of memory in your memory map. For example, the memory map shown in Example 5–2 allocates 20K words of 'C54x program memory.

Example 5–2. Sample Memory Map for the TMS320C54x Using Memory Cache Capabilities

```
MA 0,0,0x2000,R|W
MA 0x2000,0,0x2000,R|W
MA 0xc000,0,0x1000,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 are created in the root directory of that floppy drive (for example, 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 not 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.
Customizing the Memory Map

5.4 Customizing the Memory Map

The customizable 'C54x (cDSP) debugger allows you maximum flexibility in configuring a memory map. Because the size and address of the memory map is not fixed in the debugger, you can select any amount of ROM or RAM internally, externally, or both.

The following example shows how you can have both RAM and ROM mapped to the same address:

\[
\text{ma 0xc000, 0, 0x1000, R} \quad ;\text{Internal (on-chip) program ROM}
\]

\[
\text{ma 0xc000, 0, 0x1000, R|EX} \quad ;\text{External (off-chip) program ROM}
\]

During execution or when the debugger performs memory accesses, the block of memory accessed is based on the 'C54x MP/MC bit located in the PMST register. When this bit is set to 0, the on-chip program ROM is enabled. When it is set to 1, the off-chip program RAM is enabled.

The next example shows two blocks of RAM, one internal (on-chip) and one external (off-chip), mapped to the same address.

\[
\text{ma 0x0080, 0, 0x0380, R|W} \quad ;\text{Internal (on-chip) program RAM}
\]

\[
\text{ma 0x0080, 0, 0x0380, R|W|EX} \quad ;\text{External (off-chip) program RAM}
\]

For the above example, the block of memory is accessed based on the OVLY bit located in the PMST register during execution or when the debugger performs memory accesses. When this bit is set to 1, the on-chip dual-access data RAM is mapped to internal program space. When it is cleared to 0, the off-chip program RAM is enabled.

The debugger accesses the three types of memory (data, program ROM, and program RAM) according to the type of memory and the values of the MP/MC bits. The following table summarizes how the debugger accesses memory:

<table>
<thead>
<tr>
<th>Type of Memory</th>
<th>Memory Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Accesses internal memory block, then external memory block.</td>
</tr>
<tr>
<td>Program ROM</td>
<td>If MP/MC is set to 0, accesses internal memory block, then external memory block; if MP/MC is set to 1, accesses external memory block.</td>
</tr>
<tr>
<td>Program RAM</td>
<td>If OVLY is set to 1, accesses internal memory block, then external memory block; if OVLY is set to 0, accesses external memory block.</td>
</tr>
</tbody>
</table>
Customizing the Memory Map

Mapping on-chip dual-access RAM from data memory to program memory

You can configure on-chip dual-access RAM as data memory or program memory. The following steps describe how to map a block of data memory to program memory:

**Step 1:** Set OVLY (the overlay bit) in the PMST register to 1.

**Step 2:** Define the data-memory map before you define the program-memory map. It is essential to define the data-memory map for the overlay mode.

**Step 3:** Add a dummy program-memory map in the same region as the external memory. To do this, use the EX attribute for the MA command.

---

**Note:**

The sizes of the data-memory map and the program-memory map must be the same.

---

The following is an example of mapping the on-chip dual-access RAM to program memory. The example shows the commands to set the mode to overlay.

\[
\text{ma 0x0080, 1, 0x0f80, R|W|DA} \\
\text{ma 0x0080, 0, 0x0f80, R|W|EX} \\
\text{?pmst=0xffc0 ; mp/mc=0, ovly=1}
\]

Simulating data memory (ROM)

With the 'C54x simulator, you can simulate the DROM bit in the 'C541, 'C543, 'C545, or 'C546 processor. This simulation allows you to map the on-chip program memory (ROM) to the data memory. To map the program memory (ROM) to the data memory, follow these steps:

**Step 1:** Set the DROM bit (bit 3) in the PMST register to 1.

**Step 2:** Invoke the simulator with the appropriate –mv54x option.

The following example shows how to set the DROM bit to 1 from the debugger:

\[
\text{?pmst=0x08 ; DROM bit is set to 1}
\]
Customizing the Memory Map

Programming your memory

The most convenient time to set up your memory is during the initialization process. However, you can edit your memory map while your program is running.

Use the OVLY and MP/MC bits of the PMST register to set the amount of external and internal program memory you need. The values for the OVLY and MP/MC bits are as follows:

- **OVLY bit**
  - 0 = external program memory
  - 1 = internal program memory

- **MP/MC bit**
  - 0 = internal program memory (ROM)
  - 1 = external program memory

You can edit the values of the OVLY and MP/MC bits by using the debugger or by programming the PMST register. To use the debugger to edit the values of these bits, scroll down the CPU window until you see the PMST register. The CPU window is editable; you can enter the values for each bit.
Enabling Memory Mapping

5.5 Enabling Memory Mapping

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

map {on | off}

Disabling memory mapping can cause bus fault problems in the target system 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 as protected.

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

*Error in expression*
5.6 Checking the Memory Map

ml

If you want to see which memory ranges are defined, use the ML (memory list) 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, assume you issue the following MA commands:

ma 0, 0, 0x3000, ROM
ma 0x4000, 0, 0x2000, EX|RAM
ma 0, 1, 0x4000, RAM
ma 0x8000, 1, 0x2000, EX|RAM
ma 0x6, 2, 0x3, P|R

If you enter the ML command, the debugger displays the following information in the display area of the COMMAND window:

<table>
<thead>
<tr>
<th>Page</th>
<th>Memory range</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000 – 2fff</td>
<td>R</td>
</tr>
<tr>
<td>0</td>
<td>4000 – 5fff</td>
<td>R</td>
</tr>
<tr>
<td>1</td>
<td>0000 – 3fff</td>
<td>R</td>
</tr>
<tr>
<td>1</td>
<td>8000 – 9fff</td>
<td>R</td>
</tr>
<tr>
<td>2</td>
<td>0006 – 0008</td>
<td>P</td>
</tr>
</tbody>
</table>

(page 0 = program memory  page 1 = data memory  page 2 = I/O space)
5.7 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
```

- The `address` parameter 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 display area of the COMMAND window:
  ```
  Specified map not found
  ```
- The `page` parameter is a 1-digit number that identifies the type of memory (program, data, or I/O) that the range occupies:

<table>
<thead>
<tr>
<th>To identify this page,</th>
<th>Use this value as the page parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program memory</td>
<td>0</td>
</tr>
<tr>
<td>Data memory</td>
<td>1</td>
</tr>
<tr>
<td>I/O space</td>
<td>2</td>
</tr>
</tbody>
</table>

**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 (see Disconnecting an I/O port, page 5-21).

**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 5-5.
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. (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.
5.8 Using Multiple Memory Maps for Multiple Target Systems (Emulator Only)

If you are debugging multiple applications, you may need a memory map for each target system. Here is 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 use instead of the default initialization batch file.
5.9 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 P|R (input port) or P|R|W (input/output port) as the memory type. Use page 2 to simulate I/O space. 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. Use page 1 for file connects to data memory.

Connecting an I/O port

```
mc
```

The MC (memory connect) command connects P|R or P|R|W to an input or output file. MC also allows you to connect any data memory location (except 0x0000–0x001F) to an input or output file to read data from or write data into the file. The syntax for this command is:

```
mc  portaddress, page, length, filename, fileaccess
```

- **portaddress** parameter defines the address of the I/O space or data memory. This parameter can be an absolute address, any C expression, the name of a C function, or an assembly language label.

  The **portaddress** must be previously defined with the MA command (described on page 5-5) and have a keyword of either P|R (input port) or P|R|W (input/output port). The length of the address range defined for the port (or peripheral frame) can be 0x1000 to 0x1FFF bytes and does not have to be a multiple of 16.

- **page** parameter is a 1-digit number that identifies the type of memory (data or I/O) that the address occupies:

<table>
<thead>
<tr>
<th>To identify this page . . .</th>
<th>Use this value as the page parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data memory</td>
<td>1</td>
</tr>
<tr>
<td>I/O space</td>
<td>2</td>
</tr>
</tbody>
</table>

- **length** parameter defines the length of the range. This parameter can be any C expression.

- **filename** parameter can be any file name. If you connect a port or memory location to read from a file, the file must exist, or the MC command will fail.
The `fileaccess` parameter identifies the access characteristics of the I/O memory and data memory. The file access must be one of the keywords identified below:

<table>
<thead>
<tr>
<th>To identify this file access type . . .</th>
<th>Use this keyword as the <code>fileaccess</code> parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input port (I/O space)</td>
<td>P</td>
</tr>
<tr>
<td>Simulator halt at EOF of input space</td>
<td>R</td>
</tr>
<tr>
<td>(I/O space)</td>
<td></td>
</tr>
<tr>
<td>Output port (I/O space)</td>
<td>P</td>
</tr>
<tr>
<td>Read-only internal memory</td>
<td>R</td>
</tr>
<tr>
<td>Read-only external memory</td>
<td>EX</td>
</tr>
<tr>
<td>Simulator halt at EOF of input file for internal memory</td>
<td>R</td>
</tr>
<tr>
<td>Simulator halt at EOF of input file for external memory</td>
<td>EX</td>
</tr>
<tr>
<td>Write-only internal memory</td>
<td>W</td>
</tr>
<tr>
<td>Write-only external memory</td>
<td>EX</td>
</tr>
</tbody>
</table>

For I/O memory locations, the file is accessed during a read or write instruction to the associated port address. You can connect any I/O port to a file. A maximum of one input and one output file can be connected to a single port; however, multiple ports can be connected to a single file.

For data memory locations, the debugger accesses the data as follows:

- **When you are executing code:**
  - If you have specified a file, the debugger reads the data from the file and updates the memory location with that data.
  - If you have specified a file, the debugger writes the data to the memory location, as well as to the file.

- **When you are using the debugger:**
  - The debugger reads the data value from the memory location, not from the connected file.
  - If you have specified a file, the debugger writes the data to the memory location, as well as to the file.
Defining a Memory Map

If you use the NR parameter, the simulator halts execution when it reads an EOF. The debugger displays the appropriate message in the display area of the COMMAND window:

```
<addr> EOF reached - connected at port (I/O_PAGE)
```

or

```
<addr> EOF reached - connected at location (DATA_PAGE)
```

At this point, you can disconnect the file by using the MI command and attach a new file by using the MC command. If you do not do anything, the file pointer resets automatically to the beginning of the input file, and execution continues until EOF is read.

If you do not specify the NR parameter, execution does not halt, and you are not notified when EOF is reached. The file pointer resets automatically to the beginning of the input file, and the simulator resumes reading from the file.

Example 5–3 shows how input and output ports can be connected to specific memory blocks.

**Example 5–3. Connecting Input and Output Ports to Input or Output Files**

Assume that you have two data-memory blocks:

```
ma 0x100,1, 0x10, EX|RAM ;block1
ma 0x200,1, 0x10, RAM ;block2
```

- You could use the MC command to set up and connect an input file to block1:
  ```
  mc 0x100, 1, 0x1, my_input.dat, EX|R
  ```

- You could use the MC command to set up and connect an output file to block2:
  ```
  mc 0x205, 1, 0x1, my_output.dat, W
  ```

- You could use the MC command to halt simulator at EOF of input file:
  ```
  mc 0x100, 1, 0x1, my_input.dat, EX|R|NR
  ```
  or
  ```
  mc 0x100, 1, 0x1, my_input.dat, R|NR
  ```
Example 5–4 shows how to connect an input port to an input file named in.dat.

**Example 5–4. 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
.  
.  
.  
```

Use MA and MC commands to set up and connect an input port:

```
MA 0x50,2,0x1,R|P
MC 0x50,2,0x1,in.dat,R
```

Configure port address 50h as an input port.

Open file in.dat and connect it to port address 50.

Assume that the following instruction is part of your program; it reads from the file in.dat:

```
PORTR 050,data_mem
```

Read file in.dat, and put the value into the DATA_MEM location.

**Notes:**

1) You can connect a file only to configured location(s).
2) You cannot connect a file to program memory (page 0) locations.
3) You cannot connect a file to the core-memory map register area (0x0000 to 0x001F) of data memory (page 1).
4) While connecting a file to a set of locations:
   - Locations must not spread across memory block boundaries.
   - Two read-only files must not overlap.
   - Two write-only files must not overlap.
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  portaddress, page, {R | W | EX}

The portaddress and page identify the port that will be closed. The read/write/execute characteristics must match the parameter used when the port was connected.
5.10 Simulating External Interrupts (Simulator Only)

The 'C54x simulator allows you to simulate the external interrupt signals INT0 to INT3 and allows you 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 interrupt pins, INT0 to INT3 or the BIO pin.

**Note:**
The interrupt interval is expressed as a function of CPU clock cycles. Simulation begins at the first clock cycle.

**Setting up your input file**

To simulate interrupts, you must first set up an input file that lists interrupt intervals. Your file must contain a clock cycle in one of the following formats:

For the INT0, INT1, INT2, and INT3 pins, use this format:

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

For the BIO pin, you must enter the square brackets around the clock cycle and logic value. Use this format:

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

- The *clock cycle* parameter represents the CPU clock cycle in which 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
Simulating External Interrupts (Simulator Only)

This example shows three interrupts being 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.

- The logic value parameter is only for the BIO pin. You can force the signal to go high or low at specified clock cycles. 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 BIO pin to go high at the 12th cycle, low at the 23rd cycle, and high again at the 45th cycle.

- The rpt (n | EOS) parameter is optional and represents a repetition value. You can use two forms of repetition in simulating interrupts:

  - Repeat 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 the parentheses represent the portion that is repeated. Therefore, an interrupt is simulated at the 5th, 15th (5 + 10), 35th (15 + 20), 45th (35 + 10), and 65th (45 + 20) CPU clock cycles. The parameter n is a positive integer value.

  - Repeat 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, 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 the interrupt pin to your input file
- List the interrupt pins
- Disconnect an interrupt pin from a 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
```

- The *pinname* identifies the pin and must be one of the following: INT0, INT1, INT2, INT3, or BIO.
- The *filename* is the name of your input file. Make sure you have set up your input file as described in Setting up your input file on page 5-22.

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

**Example 5–5. Using the PINC Command to Connect the Input File**

Suppose you want to generate an **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 **INT2** pin.

```
pinc int2, myfile  
```

This command connects myfile to the **INT2** pin. As a result, the simulator generates an **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 is 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
>>>      
```

When you want to connect another file to an interrupt pin, the PINL command is useful for looking up an unconnected pin.

**pind**

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

```
pind  pinname
```

The *pinname* parameter identifies the interrupt pin and must be one of the following: INT0, INT1, INT2, INT3, or BIO. The PIND command detaches the file from the input pin. After executing this command, you can connect another file to the same pin.
5.11 Simulating Peripherals (Simulator Only)

With the 'C54x simulator, you can simulate the timer, a standard serial port, a buffered serial port, or a TDM serial port, depending on the device you choose to simulate. Each 'C54x device has a different set of peripherals. You can select the peripheral that you want to simulate by using the –mv option. Table 5–1 lists the option for each 'C54x device and the peripherals associated with that option/device.

Table 5–1. Debugger Options for Loading a Simulator Memory Map

<table>
<thead>
<tr>
<th>Option</th>
<th>Device Simulated</th>
<th>Peripherals Simulated</th>
</tr>
</thead>
<tbody>
<tr>
<td>–mv541 'C541</td>
<td></td>
<td>Serial port 0, serial port 1, timer</td>
</tr>
<tr>
<td>–mv542 'C542</td>
<td></td>
<td>Buffered serial port, TDM serial port, timer</td>
</tr>
<tr>
<td>–mv543 'C543</td>
<td></td>
<td>Buffered serial port, TDM serial port, timer</td>
</tr>
<tr>
<td>–mv545 'C545</td>
<td></td>
<td>Buffered serial port, serial port 1, timer</td>
</tr>
<tr>
<td>–mv546 'C546</td>
<td></td>
<td>Buffered serial port, serial port 1, timer</td>
</tr>
<tr>
<td>–mv548 'C548</td>
<td></td>
<td>2 Buffered serial ports, TDM serial port, timer, HPI</td>
</tr>
<tr>
<td>–mv545lp 'C545LP</td>
<td></td>
<td>Buffered serial port, serial port 1, timer, HPI</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type of Serial Port</th>
<th>See This Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>5.12 on page 5-27</td>
</tr>
<tr>
<td>Buffered</td>
<td>5.13 on page 5-31</td>
</tr>
<tr>
<td>TDM</td>
<td>5.14 on page 5-34</td>
</tr>
</tbody>
</table>
5.12 Simulating Standard Serial Ports (Simulator Only)

The 'C54x 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/TDRR registers, respectively.

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

The 'C54x 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 5-28) 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 these signals go high.

- The operations associated with the following memory-mapped registers:

<table>
<thead>
<tr>
<th>Register</th>
<th>Memory</th>
<th>Bits Used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPC</td>
<td>0x22</td>
<td>FO</td>
<td>Format specifier (8/16 bits)</td>
</tr>
<tr>
<td>TSPC</td>
<td>0x32</td>
<td>MCM</td>
<td>Internal/external clock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XRST/RRST</td>
<td>Transmit/receive reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XRDY/RRDY</td>
<td>Transmit/receive ready</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XSREMPY</td>
<td>Transmit register empty flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSRFULL</td>
<td>Receive register full flag</td>
</tr>
<tr>
<td>DXR</td>
<td>0x20</td>
<td>All bits are used</td>
<td>Transmit data register</td>
</tr>
<tr>
<td>TDXR</td>
<td>0x30</td>
<td>All bits are used</td>
<td>Receive data register</td>
</tr>
<tr>
<td>DRR</td>
<td>0x21</td>
<td>All bits are used</td>
<td>Receive data register</td>
</tr>
<tr>
<td>TDRR</td>
<td>0x31</td>
<td>All bits are used</td>
<td>Receive data register</td>
</tr>
</tbody>
</table>
Setting up your transmit and receive operations

The 'C54x 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 5-24). 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
```

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

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

The square brackets are used only with logic values for the BIO pin. For more information about defining clock cycles, see Section 5.10 on page 5-22.

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

```
DIVIDE r
```

The parameter `r` is a real number or integer specifying the ratio of the CPU clock rate to the serial port clock rate. 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 I/O files

Input and output files are connected to DRR/TDRR 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/TDRR register.

The input and output file formats for the standard serial port operation requires at least one line containing an 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 TDRR registers and the TFSX and TFSR pins.
Programming the simulator

To simulate the standard serial port, configure the DXR/TDXR and DRR/TDRR 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):

\begin{verbatim}
ma DRR,1,1,R\|P
ma DXR,1,1,W\|P
mc DRR,1,1,receive filename,READ
mc DXR,1,1,transmit filename,WRITE
pinc FSX,fsx timing filename
pinc FSR,fsr timing filename
\end{verbatim}

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive filename</td>
<td>The file to read data from, which simulates the input port</td>
</tr>
<tr>
<td>transmit filename</td>
<td>The file to write data to, which simulates the output port</td>
</tr>
<tr>
<td>fsx timing filename</td>
<td>The file that contains the CPU cycles at which the FSX frame synchronization pin goes high</td>
</tr>
<tr>
<td>fsr timing filename</td>
<td>The file that contains the CPU cycles at which the FSR frame synchronization pin goes high</td>
</tr>
</tbody>
</table>
5.13 Simulating Buffered Serial Ports (Simulator Only)

The 'C54x 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 also provides limited support for the simulation of the serial port control signals (frame synchronization signals) with the help of external event simulation capability. Frame synchronization signal values for receive and transmit operations at various instants of time are fed through the files associated with the pins. The 'C54x simulator supports the following operations in the buffered serial port simulation:

- **Automatic buffering and standard serial port modes**
- **Internal clocks (1/(CLKDV + 1) CPU clock) and external clocks for the transmit and receive operations.** CLKDV is the clock divide-down ratio.
- **External frame synchronization pulses** (FSX and FSR frame synchronization pulses): transmit and receive operations are initiated when these signals go high.

The operations associated with the following memory-mapped registers:

<table>
<thead>
<tr>
<th>Register</th>
<th>Memory</th>
<th>Bits Used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPC</td>
<td>0x22</td>
<td>FO, MCM, XRST/RRST, XRDY/RRDY, XSREMPY, RSRFULL</td>
<td>Format specifier (8/16 bits), Internal/external clock, Transmit/receive reset, Transmit/receive ready, Transmit register empty flag, Receive register full flag</td>
</tr>
<tr>
<td>DXR</td>
<td>0x21</td>
<td>All bits are used</td>
<td>Transmit data register</td>
</tr>
<tr>
<td>DRR</td>
<td>0x20</td>
<td>All bits are used</td>
<td>Receive data register</td>
</tr>
<tr>
<td>SPCE</td>
<td>0x23</td>
<td>CLKDV, FE, RH/TH, BXE/BRE, HALTX/HALTR</td>
<td>Clock divide-down ratio, Extended format specifier, Buffer half received or transmitted, Enable/disable automatic buffering, Switch to standalone mode after the current half is transmitted/received</td>
</tr>
<tr>
<td>AXR</td>
<td>0x38</td>
<td>All bits are used</td>
<td>Address register for transmit</td>
</tr>
<tr>
<td>ARR</td>
<td>0x3a</td>
<td>All bits are used</td>
<td>Address register for receive</td>
</tr>
<tr>
<td>BKX</td>
<td>0x39</td>
<td>All bits are used</td>
<td>Block size register for the transmit</td>
</tr>
<tr>
<td>BKR</td>
<td>0x3b</td>
<td>All bits are used</td>
<td>Block size register for the receive</td>
</tr>
</tbody>
</table>
**Setting up your transmit and receive operations**

The 'C54x 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 5-24). Use the following command syntax, selecting the appropriate command for the pin you want:

```
pinc FSX, filename
pinc FSR, filename
```

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

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

The square brackets are used only with logic values for the BIO pin. For more information about defining clock cycles, see Section 5.10 on page 5-22.

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

```
DIVIDE r
```

The parameter *r* is a real number or integer specifying the ratio of the CPU clock rate to the serial port clock rate. 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 I/O 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,R|P
ma DXR,1,1,W|P
mc DRR,1,1, receive filename ,READ
mc DXR,1,1, transmit filename ,WRITE
pinc FSX, fsx timing filename
pinc FSR, fsr timing filename
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive filename</td>
<td>The file to read data from, which simulates the input port</td>
</tr>
<tr>
<td>transmit filename</td>
<td>The file to write data to, which simulates the output port</td>
</tr>
<tr>
<td>fsx timing filename</td>
<td>The file that contains the CPU cycles at which the FSX frame synchronization pin goes high</td>
</tr>
<tr>
<td>fsr timing filename</td>
<td>The file that contains the CPU cycles at which the FSR frame synchronization pin goes high</td>
</tr>
</tbody>
</table>
5.14 Simulating TDM Serial Ports (Simulator Only)

The ‘C54x simulator supports TDM serial port transmission and reception by reading data from and writing data to the files associated with the TDXR and TDRR registers, respectively.

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

The ‘C54x 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Memory</th>
<th>Bits Used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSPC</td>
<td>0x32</td>
<td>TDM</td>
<td>Multiprocessor/normal mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MCM</td>
<td>Internal/external clock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XRST/RRST</td>
<td>Transmit/receive reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XRDY/RRDY</td>
<td>Transmit/receive ready</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XSREMPYT</td>
<td>Transmit register empty flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RSRFULL</td>
<td>Receive register full flag</td>
</tr>
<tr>
<td>TCSR</td>
<td>0x33</td>
<td>All bits are used</td>
<td>Channel select register</td>
</tr>
<tr>
<td>TRTA</td>
<td>0x34</td>
<td>All bits are used</td>
<td>Receive/transmit address register</td>
</tr>
<tr>
<td>TRAD</td>
<td>0x35</td>
<td>All bits are used</td>
<td>Receive address register</td>
</tr>
<tr>
<td>TDXR</td>
<td>0x31</td>
<td>All bits are used</td>
<td>Transmit data register</td>
</tr>
<tr>
<td>TDRR</td>
<td>0x30</td>
<td>All bits are used</td>
<td>Receive data register</td>
</tr>
</tbody>
</table>
**Setting up your transmit and receive operations**

The 'C54x 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 5-24). Use the following command syntax, selecting the appropriate command for the pin you want:

```
pinc TFSX, filename
pinc TFSR, filename
```

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

```
[clock cycle] rpt (n | EOS)
```

The square brackets are used only with logic values for the BIO pin. For more information about defining clock cycles, see Section 5.10 on page 5-22.

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

```
DIVIDE r
```

The parameter *r* is a real number or integer specifying the ratio of the CPU clock rate to the serial port clock rate. 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 I/O files

Input and output files are connected to TDRR 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 TDRR register. Use the following syntax to create the files:

channel-address data

The parameter channel-address specifies the TDM channel in which transmission/reception takes place. The parameter data specifies the value that is written or read from the file. Each field is in hexadecimal format and the fields are 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 TDRR 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 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 TDRR,1,1,R|P
ma TDXR,1,1,W|P
mc TDRR,1,1, receive filename ,READ
mc TDXR,1,1, transmit filename ,WRITE
pinc TFSX, fsx timing filename
pinc TFSR, fsr timing filename

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

**absolute clock cycle** 5-22

**addresses**
- accessible locations 5-2
- I/O address space, simulator 5-17 to 5-21
- invalid memory 5-3
- nonexistent memory locations 5-2
- protected areas 5-3, 5-12
- undefined areas 5-3, 5-12

**arrow keys**
- for HP workstations 3-10
- for SPARCstations 2-10

**assembler**
- for HP workstations 3-3
- for PC systems 1-3
- for SPARCstations 2-2

**assistance from TI** viii

**autoexec.bat file**
- environmental variables 1-6
- interfering with other applications 1-5
- invoking 1-7
- modifying 1-6
- sample 1-5

B

-- **b debugger option**
- for HP workstations 3-7
- for PC systems 1-7
- for SPARCstations 2-7

**batch files**
- autoexec.bat 1-5 to 1-7
- .cshrc
  - for HP workstations 3-6 to 3-8
  - for SPARCstations 2-6 to 2-8
- emuinit.cmd 5-16

**batch files (continued)**
- init.clr
  - for HP workstations 3-3
  - for PC systems 1-3
  - for SPARCstations 2-2
- init.cmd 5-3
- initdb.bat 1-5 to 1-7
- initialization
  - init.cmd 5-3
  - siminit.cmd 1-3, 2-2, 3-3
- mem.map 5-15
- memory map 5-15, 5-16
- mono.clr
  - for HP workstations 3-3
  - for PC systems 1-3
  - for SPARCstations 2-2, 2-3
- screen sizes
  - for HP workstations 3-3
  - for PC systems 1-3
  - for SPARCstations 2-3
- sim54x.cmd
  - for HP workstations 3-3
  - for PC systems 1-3
  - for SPARCstations 2-2
- siminit.cmd
  - for HP workstations 3-3
  - for PC systems 1-3
  - for SPARCstations 2-2
- TAKE command 5-15, 5-16

-- **bb debugger option**
- for HP workstations 3-7
- for PC systems 1-7
- for SPARCstations 2-7

-- **bl debugger option** 1-7

**BIO pseudoregister** 5-22 to 5-25
buffered serial port
  connecting I/O files  5-33
  programming the simulator  5-33
  setting up transmit and receive operations  5-32 to 5-33
  simulating  5-31 to 5-33
–bw debugger option  1-7

C

CD-ROM
  mounting
    for HP workstations  3-4
    for SPARCstations  2-4
  requirements
    for HP workstations  3-2
    for SPARCstations  2-2
  retrieving files from
    for HP workstations  3-4
    for SPARCstations  2-5
  unmounting
    for HP workstations  3-5
    for SPARCstations  2-5
CH (CHDIR) command  4-3
changes to the TMS320C5xx C Source Debugger
  User’s Guide  4-4
CHDIR (CD) command  4-3
clock cycle types  5-22
COFF
  formats accepted  4-1
  loading  5-3
  version 2, 4-1
color mapping with X Windows
  for HP workstations  3-11
  for SPARCstations  2-11
commands
  memory  5-5 to 5-16
  new for debugger  4-4
  updated for debugger  4-3
compiler
  for HP workstations  3-3
  for PC systems  1-3
  for SPARCstations  2-2
connecting an I/O port  5-17 to 5-21
connecting I/O files
  buffered serial ports  5-33
  standard serial port  5-29
connecting I/O files (continued)
  TDM serial ports  5-36
contacting Texas Instruments  viii
  .cshrc file
    for HP workstations  3-6 to 3-8
    for SPARCstations  2-6 to 2-8
current directory, changing  4-3
customizing the display
  for HP workstations  3-3, 3-11
  for PC systems  1-2, 1-3
  for SPARCstations  2-2, 2-3, 2-11

D

–d debugger option
  for HP workstations  3-7
  for SPARCstations  2-7
D_OPTIONS environment variable
  for HP workstations  3-7
  for PC systems  1-7, 2-7
D_SRC environment variable
  for HP workstations  3-7
  for PC systems  1-6
  for SPARCstations  2-7
D_DIR environment variable
  for HP workstations  3-6
  for PC systems  1-6
  for SPARCstations  2-6
DA keyword  5-5
  See also MA command
data memory
  adding to memory map  5-5
  deleting from memory map  5-14
  simulating  5-10
debugger
  displaying on a different machine 2-8, 3-8
  enhancements  4-1 to 4-4
environment setup
  for HP workstations  3-6 to 3-8
  for PC systems  1-5 to 1-7
  for SPARCstations  2-6 to 2-8
font changes
  for HP workstations  3-11
  for PC systems  1-2
  for SPARCstations  2-11
installation of software
  for HP workstations  3-4
  for PC systems  1-4
  for SPARCstations  2-4
debugger (continued)
installation verification
  for HP workstations 3-9
  for PC systems 1-8
  for SPARCstations 2-9
using with the X Window System
  for HP workstations 3-10 to 3-11
  for SPARCstations 2-10 to 2-11
using with Windows 1-1
default
memory map
  See also memory map, default
  for HP workstations 3-3
  for PC systems 1-3
  for SPARCstations 2-2
screen configuration file
  for HP workstations 3-3
  for PC systems 1-3
  for SPARCstations 2-2
defining a memory map. See memory map
DIR command 4-3
directories
auxiliary files
  for HP workstations 3-6
  for PC systems 1-6
  for SPARCstations 2-6
changing current directory 4-3
debugger software
  for HP workstations 3-4, 3-6
  for PC systems 1-4, 1-6
  for SPARCstations 2-5, 2-6
identifying additional source directories
  for HP workstations 3-7
  for PC systems 1-6
  for SPARCstations 2-7
listing contents of current directory 4-3
relative pathnames 4-3
sim54x
  for HP workstations 3-4, 3-6
  for PC systems 1-4, 1-6
  for SPARCstations 2-5, 2-6
disconnecting an I/O port 5-21
disk space requirements
  for HP workstations 3-2
  for SPARCstations 2-2
display
  color mappings on monochrome
    for HP workstations 3-11
    for SPARCstations 2-11
display (continued)
  font changes
    for HP workstations 3-11
    for PC systems 1-2
    for SPARCstations 2-11
requirements
  for HP workstations 3-2
  for PC systems 1-2
  for SPARCstations 2-2
DISPLAY environment variable
  for HP workstations 3-8
  for SPARCstations 2-8
DIVIDE command 5-28, 5-32, 5-35
divide-down ratio 5-28, 5-32, 5-35
DOS-command setup for the debugger 1-5
DROM bit in PMST rester 5-10

e
emuinit.cmd file 5-3
directed
  for HP workstations 3-10
  for SPARCstations 2-10
disk space requirements
  for HP workstations 3-2
  for SPARCstations 2-2
display
  color mappings on monochrome
    for HP workstations 3-11
    for SPARCstations 2-11
display (continued)
  font changes
    for HP workstations 3-11
    for PC systems 1-2
    for SPARCstations 2-11
requirements
  for HP workstations 3-2
  for PC systems 1-2
  for SPARCstations 2-2
DISPLAY environment variable
  for HP workstations 3-8
  for SPARCstations 2-8
DIVIDE command 5-28, 5-32, 5-35
divide-down ratio 5-28, 5-32, 5-35
DOS-command setup for the debugger 1-5
DROM bit in PMST rester 5-10

e
emuinit.cmd file 5-3
directed
  for HP workstations 3-10
  for SPARCstations 2-10
environment setup
  for HP workstations 3-6 to 3-8
  for PC systems 1-5 to 1-7
  for SPARCstations 2-6 to 2-8
evminit.cmd file 5-3
EX attribute 5-5
  See also MA command
EX|R keyword 5-18
EX|R|NR keyword 5-18
external frame synchronization signals
  buffered serial port 5-32
  standard serial port 5-28
  TDM serial port 5-35
external interrupts 5-22 to 5-25
  connecting input file 5-24
  disconnecting pins 5-25
  listing pins 5-25
  PINC command 5-24
  PIND command 5-25
  PINL command 5-25
  programming simulator 5-24 to 5-25
  setting up input files 5-22 to 5-23
  clock cycles 5-22
  repetition of a pattern 5-23
  simulating 5-22 to 5-25
EX|W keyword 5-18
### Index

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **F**  | file access keywords 5-18  
FILE command, changing the current directory 4-3  
files connecting to  
buffered serial port 5-33  
I/O port 5-17 to 5-20  
standard serial port 5-29  
TDM serial port 5-36  
disconnecting from I/O port 5-21  
font changes  
for HP workstations 3-11  
for PC systems 1-2  
for SPARCstations 2-11  
frame synchronization pins  
buffered serial port 5-32  
standard serial port 5-28  
TDM serial port 5-35  
function key mapping  
for HP workstations 3-10  
for SPARCstations 2-10 |
| **G**  | graphics card requirements, for PC systems 1-2 |
| **H**  | hardware checklist  
for HP workstations 3-2  
for PC systems 1-2  
for SPARCstations 2-2  
home key  
for HP workstations 3-10  
for SPARCstations 2-10  
host system  
for HP workstations 3-2  
for PC systems 1-2  
for SPARCstations 2-2  
HP systems  
installation  
software 3-4 to 3-5  
verifying 3-9  
requirements 3-2 to 3-3  
setting up debugger environment 3-6 to 3-8 |
| **I**  | --i debugger option  
for HP workstations 3-7  
for PC systems 1-7  
for SPARCstations 2-7  
I/O memory  
adding to memory map 5-5  
deleting from memory map 5-14  
simulating 5-17 to 5-21  
I/O port  
connecting 5-17 to 5-20  
disconnecting 5-21  
I/O space, simulating 5-17  
identifying sim54x directory. See modifying PATH statement  
identifying usable memory ranges 5-5 to 5-8  
IF/ELSE/ENDIF commands 5-3  
init.25  
for HP workstations 3-3  
for PC systems 1-3  
for SPARCstations 2-3  
init.43  
for HP workstations 3-3  
for PC systems 1-3  
for SPARCstations 2-3  
init.50  
for HP workstations 3-3  
for PC systems 1-3  
for SPARCstations 2-3  
init.clr file  
for HP workstations 3-3  
for PC systems 1-3  
for SPARCstations 2-2  
init.cmd file  
invoking 1-7  
sample 1-5  
initialization batch files  
for HP workstations 3-3  
for memory mapping 5-2 to 5-4  
for PC systems 1-3  
for SPARCstations 2-2  
init.cmd 5-3  
insert key  
for HP workstations 3-10  
for SPARCstations 2-10 |
Index

installation
software
  for HP workstations  3-4
  for PC systems   1-4
  for SPARCstations 2-4
verifying
  for HP workstations  3-9
  for PC systems   1-8
  for SPARCstations 2-9
interrupt pins   5-22 to 5-25
interrupts, simulating  5-22 to 5-25
invalid memory addresses  5-3, 5-12
invoking the simulator
  autoexec.bat file   1-7
  .cshrc file
    for HP workstations  3-8
    for SPARCstations 2-8
  initdb.bat file   1-7

K
keyboard mapping
  for HP workstations  3-10
  for SPARCstations 2-10
keysym labels
  for HP workstations  3-10
  for SPARCstations 2-10

L
LD_LIBRARY_PATH environment variable
  for HP workstations  3-8
  for SPARCstations 2-8
linker
  command files, MEMORY definition  5-2
    for HP workstations  3-3
    for PC systems   1-3
    for SPARCstations 2-2
loading, COFF files, restrictions  5-3

M
MA command   5-4, 5-5, 5-9, 5-14
MAP command   5-12
mapping keys for use with X Windows
  for HP workstations  3-10
  for SPARCstations 2-10
mapping on-chip dual-access RAM to program memory  5-10
MC command   5-17 to 5-20
MD command   5-14
MEMORY command  4-1
memory
  batch file search order  5-2 to 5-3
  commands
    MA command   5-4, 5-5, 5-9, 5-14
    MAP command   5-12
    MC command   5-17 to 5-20
    MD command   5-14
    MI command   5-21
    ML command   5-13
    MR command   5-14
data memory, simulating  5-10
default map
  for HP workstations  3-3
  for PC systems   1-3
  for SPARCstations 2-2
identifying usable ranges  5-5 to 5-8
invalid addresses  5-3
invalid locations  5-12
nonexistent locations  5-2
protected areas  5-3, 5-12
requirements, for PC systems  1-2
simulating
  I/O memory   5-17 to 5-21
  MC command   5-17 to 5-20
  MI command   5-21
undefined areas  5-3, 5-12
valid types  5-5
MEMORY definition  5-2
memory map
  adding ranges  5-5
  batch file   5-15
  checking   5-13
customizing  5-9 to 5-11
default  5-4
defining  5-2 to 5-8
deleting ranges  5-14
enabling/disabling  5-12
for HP workstations  3-3
for PC systems   1-3
for SPARCstations 2-2
listing current map  5-13
MA command   5-4, 5-5, 5-9, 5-14
MD command   5-14
ML command   5-13
modifying  5-2 to 5-14
memory map (continued)
  MR command 5-14
  multiple maps 5-16
  potential problems 5-3
  resetting 5-14
  returning to default/original 5-15
  sample 5-4
  simulating I/O ports 5-17 to 5-20, 5-21
MEMORY window 4-1
memory-cache capability 5-8
MI command 5-21
–min debugger option
  for HP workstations 3-7
  for PC systems 1-7
  for SPARCstations 2-7
ML command 5-13
modifying
  batch file (autoexec.bat) 1-5 to 1-6
  current directory 4-3
  memory map 5-2 to 5-14
PATH statement
  for HP workstations 3-6
  for PC systems 1-6
  for SPARCstations 2-6
mono.clr file
  for HP workstations 3-3
  for PC systems 1-3
  for SPARCstations 2-2
monochrome monitor color mapping with X Windows
  for HP workstations 3-11
  for SPARCstations 2-11
mounting CD-ROM
  for HP workstations 3-4
  for SPARCstations 2-4
mouse requirements
  for HP workstations 3-2
  for PC systems 1-2
  for SPARCstations 2-2
MP/MC bit in PMST register 5-9, 5-11
MR command 5-14
multiple MEMORY windows 4-1
multiple WATCH windows 4-2
–mv debugger option 4-4, 5-26
  for HP workstations 3-7
  for PC systems 1-7
  for SPARCstations 2-7

new or updated debugger commands 4-3 to 4-4
nonexistent memory locations 5-2
notational conventions iv
notes on using the MA command 5-6 to 5-7

operating system
  for HP workstations 3-3
  for PC systems 1-3
  for SPARCstations 2-2
optional files
  for HP workstations 3-3
  for PC systems 1-3
  for SPARCstations 2-2
OVLY bit in PMST register 5-9, 5-10, 5-11

page parameter
  in MA command 5-5
  in MC command 5-17
  in MD command 5-14
page-down key
  for HP workstations 3-10
  for SPARCstations 2-10
page-up key
  for HP workstations 3-10
  for SPARCstations 2-10
PATH statement
  for HP workstations 3-6
  for PC systems 1-6
  for SPARCstations 2-6
PC systems
  installation
    software 1-4
    verifying 1-8
  requirements 1-2 to 1-3
  setting up debugger environment 1-5 to 1-7
peripherals, simulating 5-26
  See also buffered serial port; standard serial port; TDM serial port
permissions
  for HP workstations 3-3
  for SPARCstations 2-2
PINC command 5-24
Index

PIND command 5-25
PINL command 5-25
port address, simulator 5-17 to 5-21
ports, simulating 5-17 to 5-20
P|R keyword 5-5, 5-18
–profile debugger option
  for HP workstations 3-7
  for PC systems 1-7
  for SPARCStations 2-7
program memory
  adding to memory map 5-5
  deleting from memory map 5-14
programming the simulator
  for simulating a buffered serial port 5-33
  for simulating a standard serial port 5-30
  for simulating a TDM serial port 5-36
  for simulating external interrupts 5-24 to 5-25
programming your memory 5-11
P|R|W keyword 5-5
P|W keyword 5-18

root privileges
  for HP workstations 3-3
  for SPARCStations 2-2
R|P|NR keyword 5-18
R|W keyword 5-5

R keyword 5-5, 5-18
RAM, on-chip dual-access, mapping 5-10
RAM|EX (R|W|EX) keyword 5-5
read-access conflict 5-3
receive operation
  buffered serial port simulation 5-32
  standard serial port simulation 5-28
  TDM serial port simulation 5-35
reinitializing the shell
  for HP workstations 2-8
  for SPARCStations 3-8
related documentation v to vi
relative clock cycle 5-22
relative pathnames 4-3
repetition in simulating interrupts 5-23
requirements. See hardware checklist; software checklist
retrieving files from CD-ROM
  for HP workstations 3-4
  for SPARCStations 2-5
R|NR keyword 5-18

shell, reinitializing
  for HP workstations 3-8
  for SPARCStations 2-8
sim54x
  command options
    for HP workstations 3-7
    for PC systems 1-7, 2-7
  directory
    for HP workstations 3-4
    for PC systems 1-4, 1-6
    for SPARCStations 2-5
  verifying the software installation
    for HP workstations 3-9
    for PC systems 1-8
    for SPARCStations 2-9
sim54x directory
  for HP workstations 3-6
  for PC systems 1-6
  for SPARCStations 2-6

S
–s debugger option
  for HP workstations 3-7
  for PC systems 1-7
  for SPARCStations 2-7
SA keyword 5-5
SAFEHALT command 4-4
sample batch file 5-4
sample memory maps 5-4, 5-8
serial ports
  programming
    buffered 5-33
    standard 5-30
    TDM 5-36
  simulating
    buffered 5-31 to 5-33
    standard 5-27 to 5-30
    TDM 5-34 to 5-36
setting up transmit and receive operations
  buffered serial port 5-32
  standard serial port 5-28
  TDM serial port 5-35

Index-7
sim54x.cmd file
  for HP workstations 3-3
  for PC systems 1-3
  for SPARCstations 2-2

sim54xw.exe 1-4

siminit.cmd file
  for HP workstations 3-3, 5-3
  for PC systems 1-3, 5-3
  for SPARCstations 2-2, 5-3

simulating
  buffered serial port 5-31 to 5-33
  data memory 5-10
  I/O space 5-17
  interrupts 5-22 to 5-25
  peripherals 5-26 to 5-34
  standard serial port 5-27 to 5-30
  TDM serial port 5-34 to 5-36

simulator
  enhancements 4-1 to 4-4
  environment setup
    for HP workstations 3-6 to 3-8
    for PC systems 1-5 to 1-7
    for SPARCstations 2-6 to 2-8
  I/O memory 5-17 to 5-21
  installation of software
    for HP workstations 3-4
    for PC systems 1-4
    for SPARCstations 2-4
  installation verification
    for HP workstations 3-9
    for PC systems 1-8
    for SPARCstations 2-9
  programming
    buffered serial port 5-33
    external interrupts 5-24 to 5-25
    standard serial ports 5-30
    TDM serial port 5-36

software checklist
  for HP workstations 3-3
  for PC systems 1-3
  for SPARCstations 2-2

SPARCstations
  installation
    software 2-4 to 2-5
    verifying 2-9
  requirements 2-2 to 2-3
  setting up debugger environment 2-6 to 2-8

special keys
  for HP workstations 3-10
  for SPARCstations 2-10

standard serial port
  connecting I/O files 5-29 to 5-30
  programming the simulator 5-30
  setting up transmit and receive operations 5-28
  simulating 5-27 to 5-30

synchronization, external frame
  buffered serial port 5-32
  standard serial port 5-28
  TDM serial port 5-35

system commands
  CD command 4-3
  DIR command 4-3
  SAFEHALT command 4-4
  TAKE command 5-15

system requirements. See hardware checklist; software checklist

T

d–t debugger option
  during debugger invocation 5-2
  for HP workstations 3-7
  for PC systems 1-7
  for SPARCstations 2-7
  TAKE command 5-15
  reading new memory map 5-16
  target system, SAFEHALT command 4-4
  TDM serial port
    connecting I/O files 5-36
    programming the simulator 5-36
    setting up transmit and receive operations 5-35
    simulating 5-34 to 5-36
  technical support viii
  transmit operation
    buffered serial port simulation 5-32
    standard serial port simulation 5-28
    TDM serial port simulation 5-35

U

utilities
  xev
    for HP workstations 3-10
    for SPARCstations 2-10

Index-8
utilities (continued)
  xmodmap
    for HP workstations  3-10
    for SPARCstations  2-10
  xrdb
    for HP workstations  3-11
    for SPARCstations  2-11

V
–v debugger option
  for HP workstations  3-7
  for PC systems  1-7
  for SPARCstations  2-7
verifying the software installation
  for HP workstations  3-9
  for PC systems  1-8
  for SPARCstations  2-9

W
W keyword  5-5, 5-18
WA command  4-2
WATCH window  4-2
WD command  4-2
window name parameter
  MEMORY window  4-1
  WATCH window  4-2
Windows systems. See PC systems
WR command  4-2

X
–x debugger option
  for HP workstations  3-7
  for PC systems  1-7
  for SPARCstations  2-7
X Window System
  displaying debugger on a different machine  2-8, 3-8
    for HP workstations  3-10 to 3-11
    for SPARCstations  2-10 to 2-11
.Xdefaults file
  for HP workstations  3-11
  for SPARCstations  2-11
xev utility
  for HP workstations  3-10
  for SPARCstations  2-10
xmodmap utility
  for HP workstations  3-10
  for SPARCstations  2-10
xrdb utility
  for HP workstations  3-11
  for SPARCstations  2-11