TMS320C54x Code Generation Tools Getting Started Guide

Release 1.20

Literature Number: SPRU147B
Manufacturing Part Number: 2617679-9741 revision D
March 1997

Printed on Recycled Paper

IMPORTANT NOTICE

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

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

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

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

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

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Preface

Read This First

About This Manual

The *TMS320C54x Code Generation Tools Getting Started Guide* tells you how to install release 1.20 of the TMS320C54x code generation tools on your system. It also does the following:

- Tells you how to set environment variables for parameters that you use often
- Gets you started using the compiler, linker, and assembler
- Details the enhancements included in this release of the code generation tools
- Describes how to resolve problems that you may encounter on a PC™ running MS-DOS™ or OS/2™

Notational Conventions

This document uses the following conventions.

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

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

  `cd /cdrom/hp`

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

  mkdir *tool_dir*

  mkdir is the command. This command has one parameter, indicated by *tool_dir*.

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

  set DOS4GVM=[*option***:***value*] [*option***:***value*] ...

  This command allows you to specify one or more options and to indicate values with each option, if appropriate.

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

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

  `{ * | *+ | *– }`

  This provides three choices: `*`, `*+`, or `*–`.

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

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.

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.

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.

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.

TMS320 Third-Party Support Reference Guide (literature number SPRU052) alphabetically lists over 100 third parties that provide various products that serve the family of '320 digital signal processors. A myriad of products and applications are offered—software and hardware development tools, speech recognition, image processing, noise cancellation, modems, etc.

Digital Signal Processing Applications with the TMS320 Family, Volumes 1, 2, and 3 (literature numbers SPRA012, SPRA016, SPRA017) Volumes 1 and 2 cover applications using the 'C10 and 'C20 families of fixed-point processors. Volume 3 documents applications using both fixed-point processors as well as the 'C30 floating-point processor.

Related Documentation

You can use the following books to supplement this user's guide:

Advanced C: Techniques and Applications, Sobelman, Gerald E., and David E. Krekelberg, Que Corporation

American National Standard for Information Systems—Programming Language C X3.159-1989, American National Standards Institute (ANSI standard for C)

Programming in C, Kochan, Steve G., Hayden Book Company

The C Programming Language (second edition), by Brian W. Kernighan and Dennis M. Ritchie, published by Prentice-Hall, Englewood Cliffs, New Jersey, 1988

Understanding and Using COFF, Gircys, Gintaras R., published by O'Reilly and Associates, Inc.

Trademarks

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

MS-DOS is a registered trademark of Microsoft Corporation.

OS/2 is a trademark of International Business Machines Corporation.

Solaris is a trademark of Sun Microsystems, Inc.

SPARC is a trademark of SPARC International, Inc., but 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.

Windows is a registered trademark of Microsoft Corporation.

XDS510 is a trademark of Texas Instruments Incorporated.

If You Need Assistance . . .

❑ World-Wide Web Sites

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

❑ North America, South America, Central America

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

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

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:

Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32	Email: epic@ti.com
Deutsch	+49 8161 80 33 11 or +33 1 30 70 11 68		
English	+33 1 30 70 11 65		
Francais	+33 1 30 70 11 64		
Italiano	+33 1 30 70 11 67		
EPIC Modem BBS	+33 1 30 70 11 99		
European Factory Repair	+33 4 93 22 25 40		
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10	

❑ Asia-Pacific

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

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

❑ Japan

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

❑ Documentation

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

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

Email: comments@books.sc.ti.com

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

Contents

1 Setting Up the Tools on a PC Running DOS or Windows 3.1 1-1

Provides installation instructions for PCs running MS-DOS or Windows 3.1.

1.1 System Requirements 1-2
1.2 Installing the Tools 1-3
1.3 Setting Up the Environment 1-4
1.4 Performance Considerations 1-11
1.5 Where to Go From Here 1-12

2 Setting Up the Tools on a PC Running OS/2 2-1

Provides installation instructions for PCs running OS/2.

2.1 System Requirements 2-2
2.2 Installing the Tools 2-3
2.3 Setting Up the Environment 2-4
2.4 Where to Go From Here 2-8

3 Setting Up the Tools on a SPARCstation 3-1

Provides installation instructions for SPARCstations running SunOS.

3.1 System Requirements 3-2
3.2 Mounting the CD-ROM and Installing the Tools 3-3
3.3 Setting Up the Environment 3-5
3.4 Where to Go From Here 3-9

4 Setting Up the Tools on an HP Workstation 4-1

Provides installation instructions for HP 9000 Series 700 computers running with HP-UX.

4.1 System Requirements 4-2
4.2 Mounting the CD-ROM and Installing the Tools 4-3
4.3 Setting Up the Environment 4-4
4.4 Where to Go From Here 4-8

5 Getting Started With the Code Generation Tools 5-1

Provides an overview of how to invoke and use assembler, linker, and compiler.

5.1 Getting Started With the Assembler and Linker 5-2
5.2 Getting Started With the C Compiler 5-6

6 Release Notes 6-1
Describes the enhancements and fixed problems for this release.

6.1 ’C54x Silicon Problem Workaround 6-2
6.2 Release Enhancements 6-3
6.3 Fixed Problems 6-6
6.4 Undocumented Features 6-10

A Troubleshooting DOS Systems A-1
Lists kernel and DOS/4G error messages and explains how you can resolve them.

A.1 Troubleshooting in the Protected-Mode Environment A-2
A.2 Kernel Error Messages A-5
A.3 DOS/4G Error Messages A-9

B Glossary B-1
Defines acronyms and key terms used in this book.

Chapter 1

Setting Up the Tools on a PC Running DOS or Windows 3.1

This chapter helps you install release 1.20 of the TMS320C54x code generation tools and set up your code-development environment on a PC running MS-DOS or Windows™ 3.1. These tools include an optimizing C compiler and a full set of assembly language tools for developing and manipulating assembly language and object (executable) code.

The C compiler tools include:

- ❑ Parser
- ❑ Optimizer
- ❑ Code generator
- ❑ Interlist utility
- ❑ Library-build utility

The assembly language tools are composed of the following:

- ❑ Assembler
- ❑ Archiver
- ❑ Linker
- ❑ Absolute lister
- ❑ Cross-reference lister
- ❑ Hex conversion utility
- ❑ Mnemonic to algebraic assembly code translator (MS-DOS only)

Release 1.20 supports extended memory on DOS. Extended memory lets you compile or assemble large files that could not be built previously under DOS. Extended memory is enabled by the DOS/4GW™ memory extender from Tenberry Software, Inc. (formerly Rational Systems, Inc.), which is embedded in the TMS320C54x code generation tools package.

Topic **Page**

1.1 System Requirements 1-2
1.2 Installing the Tools 1-3
1.3 Setting Up the Environment 1-4
1.4 Performance Considerations 1-11
1.5 Where to Go From Here 1-12

1.1 System Requirements

To install the TMS320C54x code generation tools on a PC, you need the following items:

- ❑ 80386-, 80486-, or Pentium-based PC running MS-DOS
- ❑ 4–16 Mbytes of free memory. 16 Mbytes is recommended to ensure optimum performance when compiling large C functions
- ❑ 10 Mbytes of disk space for the executable files and libraries
- ❑ CD-ROM drive

Note: Memory Needed

16 Mbytes of available memory is recommended when installing the code generation tools on a PC. You can have a minimum of 4 Mbytes of memory, but you can expect some performance problems. You may want to free as much memory as possible before installing the tools, especially if you have less than 16 Mbytes.

1.2 Installing the Tools

The code generation tools package is shipped on CD-ROM. The installation instructions vary according to your operating system.

Installing the tools on DOS systems

To install the tools on a DOS system, follow these steps:

1) Insert the TMS320C54x Software Toolkit CD-ROM into your CD-ROM drive.

2) Change to the CD-ROM drive (replace d with the name of your CD-ROM drive):

   `d:` ↵

3) Enter the following command:

   `install` ↵

4) Follow the on-screen instructions.

Installing the tools on Windows 3.1 systems

To install the tools on a Windows 3.1 system, perform the following steps:

1) Insert the TMS320C54x Software Toolkit CD-ROM into your CD-ROM drive.

2) Start Windows 3.1.

3) From the File menu, select Run.

4) In the dialog box, enter the following command (replace d with the name of your CD-ROM drive):

   `d:\setup.exe`

5) Click on OK.

6) Follow the on-screen instructions.

1.3 Setting Up the Environment

Before or after you install the tools, you can define *environment variables* that set certain code generation tool parameters you normally use. An environment variable is a system symbol that you define and assign to a string. When you use environment variables, default values are set, making each individual invocation of the tools simpler because these parameters are automatically specified. When you invoke a tool, you can use command-line options to override many of the defaults that are set with environment variables.

The code generation tools use environment variables for finding or obtaining certain types of information. By default, the installation program sets up these environment variables:

```
set PATH=C:\tool_dir;%PATH%
set A_DIR=C:\tool_dir
set C_DIR=C:\tool_dir
set DOS4GVM=VirtualMemory:ON
set DOS4G=
```

If you choose not to have the environment variables set up automatically, you can modify your autoexec.bat file to include the SET commands above.

The remainder of this section describes these environment variables and other variables that you can define.

Identifying the directory that contains the executable files (PATH statement)

You must include the *tool_dir* directory in your PATH statement. This allows you to specify the assembler and compiler tools without specifying the name of the directory that contains the executable files.

Modify your autoexec.bat file to change the path information, and add the following to the end of the PATH statement:

;C:*tool_dir*

Identifying alternate directories for the assembler (A_DIR)

By default, the assembler searches for copy/include files or macro libraries in the current directory and then in directories named by the –i (name alternate directories) option. Use the A_DIR environment variable to define additional search paths. The format of the command for assigning the environment variable is as follows:

set A_DIR=*pathname1*[*; pathname 2 . . .*]

You can separate the pathnames with a semicolon or a blank.

Once you set A_DIR, you can use the .copy, .include, or .mlib directive in assembly source without specifying path information.

For more information on the –i option, see the *TMS320C54x Assembly Language Tools User's Guide* or the *TMS320C54x Optimizing C Compiler User's Guide*.

Identifying alternate directories for the compiler (C_DIR)

By default, the compiler searches the current directory for #include files and object libraries such as the runtime-support and C I/O libraries. Use the C_DIR environment variable to define additional search paths. The format of the command for assigning the environment variable is as follows:

set C_DIR=*pathname1*[*;pathname 2 . . .*]

You can separate the pathnames with a semicolon or a blank.

Once you set C_DIR, you can use the #include directive in your C source code without specifying path information.

Setting default shell options (C_OPTION)

When using the shell program (cl500), you might find it useful to set default options for the compiler, assembler, and linker using the C_OPTION environment variable. After reading the command line options and the input filenames, the shell reads the contents of the C_OPTION environment variable. The options and/or input filenames that you define with C_OPTION are used every time you run the shell.

Setting up default options with the C_OPTION environment variable is especially useful when you want to run consecutive times with the same set of options and/or input files.

The syntax for the C_OPTION environment variable is:

set C_OPTION=*option1*[*option2 . . .*]

Options specified in the environment variable are specified in the same way and have the same meaning as they do on the command line. For more information about compiler and assembler options, see the *TMS320C54x Optimizing C Compiler User's Guide* and the *TMS320C54x Assembly Language Tools User's Guide*.

For example, if you want to always run quietly (the –q option), enable C source interlisting (the –s option), and link (the –z option), set up the C_OPTION environment variable as follows:

```
set C_OPTION=-qs -z
```

In this example, each time you run the shell program, it runs the linker. Any options following –z on the command line or in C_OPTION are passed to the linker. This enables you to use the C_OPTION environment variable to specify default compiler and linker options and then specify additional compiler and linker options on the shell command line. If you have set –z in the environment variable and want to compile only, use the –c option of the shell. These additional examples assume C_OPTION is set as shown above:

```
cl500  *.c               ; compiles and links
cl500 -c *.c             ; only compiles
cl500 *.c -z lnk.cmd     ; compiles and links using a
                         ; command file
cl500 -c *.c -z lnk.cmd  ; only compiles (-c overrides -z)
```

For more information on linker options, see the *Linker Options* section in the *TMS320C54x Assembly Language Tools User's Guide.*

Using virtual memory (DOS4GVM)

Virtual-memory management (VMM) allows protected-mode programs to use more RAM than your computer actually has. The DOS4GVM environment variable controls VMM. You can set the DOS4GVM environment variable using the following format:

set DOS4GVM=[*option*:*value*] [*option*:*value*] ...

You must use a colon before the value for each option; the DOS command shell does not accept an equal sign in place of the colon. Unless you specify otherwise for your system, these options are not case sensitive.

DOS4GVM options take effect only when VMM is enabled, which causes the default values to be used for all options. To enable VMM, enter:

```
set DOS4GVM=VirtualMemory:ON
```

The DOS4GVM options are described below:

DeleteSwapFile **DeleteSwapFile:**{**ON**|**OFF**}

By default, a new swap file is created each time your code runs, which slows down program startup. If you run your code over and over again and you can spare the disk space, you can set DeleteSwapFile to OFF so that an existing swap file is reused. If your code spawns another program that uses extended memory, do not set DeleteSwapFile.

PhysMax **PhysMax:***n*{**K**|**M**|**G**}

PhysMax specifies the maximum amount of physical memory (RAM) managed by VMM. The default is all available memory up to 64 Mbytes. This setting minimizes disk swapping so that large programs run as fast as possible. However, the default setting might actually slow down small programs on machines with a lot of memory because more memory is managed than is actually needed.

You might want to restrict the amount of physical memory VMM uses for the following reasons:

- Your code is small, and you know its maximum memory requirement. You can speed up the start-up by telling VMM not to manage everything.
- You need to spawn another program that uses extended memory, and you need to leave enough memory available for the other program.

PhysMin **PhysMin:***n*{**K**|**M**|**G**}

PhysMin specifies the minimum amount of physical memory (RAM) managed by VMM. The default is 1024 Kbytes. Set PhysMin to the minimum hardware requirement necessary for running your code. If your code requires a 4-Mbyte machine, set PhysMin to 4 Mbytes. If your code is small and can run in 512 Kbytes, set PhysMin to 512 Kbytes.

SwapFileName **SwapFileName:**[*path*][*filename*]

SwapFileName specifies the filename of the swap file. The default filename is DOS4GVM.SWP, and the file is placed in the directory where the executable file resides. Specify the complete path name if you want to keep the swap file in another directory.

SwapInc **SwapInc:***n*{**K**|**M**|**G**}

SwapInc specifies the size by which the swap file grows. The default size is 4096 Kbytes. As your program runs, the swap file increases by 4 Mbytes when the file needs to grow. A smaller size causes the swap file to take less time to increase but to increase more frequently. A larger size causes the swap file to take longer to increase but to increase less frequently.

To have a static swap file rather than a dynamic swap file, set SwapInc to 0 and set SwapMin to the static size you want.

SwapMin **SwapMin:***n*{**K**|**M**|**G**}

SwapMin specifies the minimum or initial size of the swap file. The default is 0 bytes. If you want VMM to create a full-size swap file at startup time, set SwapMin to the full size of the swap file and set SwapInc to 0.

VirtualSize **VirtualSize:***n*{**K**|**M**|**G**}

VirtualSize specifies the size of the virtual memory space. The default is 16384 Kbytes. Set VirtualSize to a larger size if your program uses more than 16 Mbytes of code and data, but do not set it to more than twice the size of your program's memory requirement.

You can change the defaults in two ways:

1) Specify parameter values as arguments to the DOS4GVM environment variable, as shown in the example below. Note that you must have at least 8192 Kbytes of free memory to use this example:

```
set DOS4GVM=deleteswapfile:ON physmax:8192K swapfilename:c:\swap.tmp
```

2) Create a configuration file with the filetype extension .VMC, and call that file as an argument to the DOS4GVM environment variable, as shown below:

   ```
   set DOS4GVM=@NEW4G.VMC
   ```

   A .VMC file contains VMM parameters and settings, as shown in the example below. You can include comments. Comments on lines by themselves must be preceded by an exclamation point (!). Comments that follow option settings must be preceded by white space. Do not insert blank lines; processing stops at the first blank line.

   ```
   !Sample .VMC file
   !This file shows the default parameter values
   physmin:512K        At least 512 bytes of RAM required
   physmax:4M          Uses no more than 4MB of RAM
   virtualsize:16M     Swap file + allocated memory is 16MB
   ```

See Appendix A, *Troubleshooting DOS Systems*, for information on problems and solutions when using DOS/4GW.

Specifying a temporary file directory (TMP)

The shell program creates intermediate files as it processes your program. For example, the parser phase of the shell creates a temporary file used as input by the code generator phase. By default, the shell puts intermediate files in the current directory. However, you can name a specific directory for temporary files by using the TMP environment variable.

This feature allows the use of a RAM disk or other file systems. It also allows source files to be compiled from a remote directory without writing any files into the directory where the source resides; this is useful for protected directories.

To set the TMP environment variable, use this syntax:

set TMP=*pathname*

For example, to set up a directory named temp for intermediate files on your hard drive, enter:

```
set TMP=C:\temp
```

Removing indicator of real or protected mode (C_MODE)

Previous versions of the TMS320C54x code generation tools used C_MODE to indicate whether to use the real- or protected-mode tools. *This environment variable is no longer recognized and is ignored*. You can safely delete C_MODE from your environment if you had it set for use with a DOS version of the TMS320C54x code generation tools prior to version 1.00.

Resetting defined environment variables

The environment variables that you define remain set until you reboot the system. If you want to clear an environment variable, use this command:

set *variable name***=**

For example, to reset the A_DIR environment variable, enter:

```
set A_DIR=
```

1.4 Performance Considerations

You may notice a speed degradation when you use the code generation tools. Much of this speed degradation is due to the switch rate from protected to real mode necessitated by DOS calls. Higher-speed processors and later-generation processors in the 80386, 80486, and Pentium series minimize the time needed for this switch.

Virtual-memory management (VMM) may also degrade system performance. It is recommended that VMM be enabled only for programs that cannot be built with VMM disabled.

If you encounter error messages when you use the code generation tools on a PC with DOS, run PMINFO to determine the configuration of your system before you contact technical support. For more information about PMINFO, see Appendix A, *Troubleshooting DOS Systems*.

1.5 Where to Go From Here

Your code generation tools are now installed. At this point, you should do the following:

- ❑ Go to Chapter 5, *Getting Started With the Code Generation Tools*. This chapter provides you with an overview of how to invoke and use the assembler, linker, and compiler.
- ❑ Read Chapter 6, *Release Notes*, to understand the new features included in the 1.20 release of the code generation tools.
- ❑ Use Appendix A, *Troubleshooting DOS Systems*, as necessary. This appendix lists kernel and DOS/4G error messages and explains how you can resolve the messages.

Chapter 2

Setting Up the Tools on a PC Running OS/2

This chapter helps you install release 1.20 of the TMS320C54x code generation tools and set up your code-development environment on a PC running OS/2™. These tools include an optimizing C compiler and a full set of assembly language tools for developing and manipulating assembly language and object (executable) code.

The C compiler tools include:

- ❑ Parser
- ❑ Optimizer
- ❑ Code generator
- ❑ Interlist utility
- ❑ Library-build utility

The assembly language tools are composed of the following:

- ❑ Assembler
- ❑ Archiver
- ❑ Linker
- ❑ Absolute lister
- ❑ Cross-reference lister
- ❑ Hex conversion utility

Topic **Page**

2.1 System Requirements 2-2
2.2 Installing the Tools 2-3
2.3 Setting Up the Environment 2-4
2.4 Where to Go From Here 2-8

2.1 System Requirements

To install the TMS320C54x code generation tools on a PC, you need the following items:

- ❑ 80386-, 80486-, or Pentium-based PC running OS/2 version 2.x or OS/2 Warp
- ❑ 4–16 Mbytes of free memory. 16 Mbytes is recommended to ensure optimum performance when compiling large C functions
- ❑ 10 Mbytes of disk space for the executable files and libraries
- ❑ CD-ROM drive

Note: Memory Needed

16 Mbytes of available memory is recommended when installing the code generation tools on a PC. You can have a minimum of 4 Mbytes of available memory, but you can expect performance problems. You may want to free as much memory as possible before installing the tools, especially if you have less than 16 Mbytes.

2.2 Installing the Tools

The code generation tools package is shipped on CD-ROM. To install the tools on a PC running OS/2, follow these steps:

1) Start a DOS-Window session.

2) Change to the CD-ROM drive (replace d with the name of your CD-ROM drive):

   `d:` ↵

3) Enter the following command:

   `install` ↵

4) Follow the on-screen instructions.

2.3 Setting Up the Environment

Before or after you install the tools, you can define *environment variables* that set certain code generation tool parameters you normally use. An environment variable is a system symbol that you define and assign to a string. When you use environment variables, default values are set, making each individual invocation of the tools simpler because these parameters are automatically specified. When you invoke a tool, you can use command-line options to override many of the defaults that are set with environment variables.

The code generation tools use environment variables for finding or obtaining certain types of information. By default, the installation program sets up these environment variables:

```
set PATH=C:\tool_dir;%PATH%
set A_DIR=C:\tool_dir
set C_DIR=C:\tool_dir
```

If you choose not to have the environment variables set up automatically, you can modify your config.sys file to include the SET commands above.

The remainder of this section describes these environment variables and other variables that you can define.

Identifying the directory that contains the executable files (PATH statement)

You must include the *tool_dir* directory in your PATH statement. This allows you to specify the assembler and compiler tools without specifying the name of the directory that contains the executable files.

- ❏ If you modify your config.sys file to change the path information, add the following to the end of the PATH statement:

  ;C:*tool_dir*

- ❏ If you set the PATH statement from the command line, use this format:

  set PATH=C:*tool_dir***;%PATH%**

  The addition of **;%PATH%** ensures that this PATH statement does not undo the PATH statements in any other batch files (including the autoexec.bat file).

Identifying alternate directories for the assembler (A_DIR)

By default, the assembler searches for copy/include files or macro libraries in the current directory and then in directories named by the –i (name alternate directories) option. Use the A_DIR environment variable to define additional search paths. The format of the command for assigning the environment variable is as follows:

set A_DIR=*pathname1*[*; pathname 2 . . .*]

You can separate the pathnames with a semicolon or a blank.

Once you set A_DIR, you can use the .copy, .include, or .mlib directive in assembly source without specifying path information.

For more information on the –i option, see the *TMS320C54x Assembly Language Tools User's Guide* or the *TMS320C54x Optimizing C Compiler User's Guide*.

Identifying alternate directories for the compiler (C_DIR)

By default, the compiler searches the current directory for #include files and object libraries such as the runtime-support and C I/O libraries. Use the C_DIR environment variable to define additional search paths. The format of the command for assigning the environment variable is as follows:

set C_DIR=*pathname1*[*; pathname 2 . . .*]

You can separate the pathnames with a semicolon or with a blank.

Once you set C_DIR, you can use the #include directive in your C source code without specifying path information.

Setting default shell options (C_OPTION)

When using the shell program (cl500), you might find it useful to set default options for the compiler, assembler, and linker using the C_OPTION environment variable. After reading the command line options and the input filenames, the shell reads the contents of the C_OPTION environment variable. The options and/or input filenames that you define with C_OPTION are used every time you run the shell.

Setting up default options with the C_OPTION environment variable is especially useful when you want to run consecutive times with the same set of options and/or input files.

The syntax for the C_OPTION environment variable is:

set C_OPTION=*option1*[*option2 . . .*]

Options specified in the environment variable are specified in the same way and have the same meaning as they do on the command line. For more information about compiler and assembler options, see the *TMS320C54x Optimizing C Compiler User's Guide* and the *TMS320C54x Assembly Language Tools User's Guide*.

For example, if you want to always run quietly (the –q option), enable C source interlisting (the –s option), and link (the –z option), set up the C_OPTION environment variable as follows:

```
set C_OPTION=-qs -z
```

In this example, each time you run the shell program, it runs the linker. Any options following –z on the command line or in C_OPTION are passed to the linker. This enables you to use the C_OPTION environment variable to specify default compiler and linker options and then specify additional compiler and linker options on the shell command line. If you have set –z in the environment variable and want to compile only, use the –c option of the shell. These additional examples assume C_OPTION is set as shown above:

```
cl500  *.c               ; compiles and links
cl500  -c *.c            ; only compiles
cl500  *.c -z lnk.cmd    ; compiles and links using a
cl500                    ; command file
cl500  -c *.c -z lnk.cmd ; only compiles (-c overrides -z)
```

For more information on linker options, see the *Linker Options* section in the *TMS320C54x Assembly Language Tools User's Guide.*

Specifying a temporary file directory (TMP)

The shell program creates intermediate files as it processes your program. For example, the parser phase of the shell creates a temporary file used as input by the code generator phase. By default, the shell puts intermediate files in the current directory. However, you can name a specific directory for temporary files by using the TMP environment variable.

This feature allows use of a RAM disk or other file systems. It also allows source files to be compiled from a remote directory without writing any files into the directory where the source resides; this is useful for protected directories.

To set the TMP environment variable, use this syntax:

set TMP=*pathname*

For example, to set up a directory named temp for intermediate files on your hard drive, enter:

```
set TMP=C:\temp
```

Removing indicator of real or protected mode (C_MODE)

Previous versions of the TMS320C54x code generation tools used C_MODE to indicate whether to use the real- or protected-mode tools. *This environment variable is no longer recognized and will be ignored*. You can safely delete C_MODE from your environment if you had it set for use with a DOS version of the TMS320C54x code generation tools prior to version 1.00.

Resetting defined environment variables

The environment variables that you define remain set until you reboot the system. If you want to clear an environment variable, use this command:

set *variable name***=**

For example, to reset the A_DIR environment variable, enter:

```
set A_DIR=
```

2.4 Where to Go From Here

Your code generation tools are now installed. At this point, you should do the following:

- ❑ Go to Chapter 5, *Getting Started With the Code Generation Tools*. This chapter provides you with an overview of how to invoke and use the assembler, linker, and compiler.
- ❑ Read Chapter 6, *Release Notes*, to understand the new features included in the 1.20 release of the code generation tools.

Chapter 3

Setting Up the Tools on a SPARCstation

This chapter helps you install release 1.20 of the TMS320C54x code generation tools and set up your code-development environment on a SPARCstation running SunOS™ versions 4.1.3 (or higher) or SunOS 5.x (also known as Solaris 2.x). These tools include an optimizing C compiler and a full set of assembly language tools for developing and manipulating assembly language and object (executable) code.

The C compiler tools include:

- Parser
- Optimizer
- Code generator
- Interlist utility
- Library-build utility

The assembly language tools are composed of the following:

- Assembler
- Archiver
- Linker
- Absolute lister
- Cross-reference lister
- Hex conversion utility
- Mnemonic to algebraic assembly code translator

This release is dynamically linked to take advantage of shared libraries.

Topic **Page**

3.1	**System Requirements**	**3-2**
3.2	**Mounting the CD-ROM and Installing the Tools**	**3-3**
3.3	**Setting Up the Environment**	**3-5**
3.4	**Where to Go From Here**	**3-9**

3.1 System Requirements

To install the TMS320C54x code generation tools on a SPARCstation, you need the following items:

- ❑ SPARCstation or compatible system with SPARCstation 2 class or higher performance
- ❑ 4 Mbytes of disk space for the software tools
- ❑ OpenWindows™ version 3.0 (or higher) running under SunOS version 4.1.3 (or higher). If you are using SunOS 5.x (also known as Solaris 2.x), you must have the Binary Compatibility Package (BCP) installed; if you do not, get your system administrator's help.
- ❑ CD-ROM drive
- ❑ Root privileges to mount and unmount the CD-ROM if you have SunOS 4.1.3, SunOS 5.0, or SunOS 5.1

3.2 Mounting the CD-ROM and Installing the Tools

To install the software tools, you must mount the CD-ROM, copy the files to your system, and unmount the CD-ROM.

Mounting the CD-ROM

The code generation tools package is shipped on CD-ROM. The steps to mount the CD-ROM vary according to your operating system version:

- If you have a SunOS 4.1.3, as root, load the CD-ROM into the drive and enter the following from a command shell:

  ```
  # mount –rt hsfs /dev/sr0 /cdrom ↵
  # exit ↵
  % cd /cdrom/sparc ↵
  ```

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

  ```
  # mount –rF hsfs /dev/sr0 /cdrom ↵
  # exit ↵
  % cd /cdrom/cdrom0/sparc ↵
  ```

- If you have SunOS 5.2 (Solaris 2.2) or higher:
  - If the CD-ROM drive is already attached, load the CD-ROM into the drive. Enter:

    ```
    % cd /cdrom/cdrom0/sparc ↵
    ```

  - If the CD-ROM drive is not attached, you must shut down your system to the PROM level (at the OK prompt) and attach the CD-ROM drive. As root, enter:

    ```
    # boot –r ↵
    ```

    Log on, load the CD-ROM into the drive, and enter:

    ```
    % cd /cdrom/cdrom0/sparc ↵
    ```

Installing the tools

Be sure you are not logged on as root. To install the software tools, follow these steps:

1) If you don't already have a tools directory, create one. Enter:

   mkdir *tool_dir* ↵

   Replace *tool_dir* with your own directory name, including the path information, to install the tools.

2) Copy the files to your directory:

   cp –r * *tool_dir* ↵

Unmounting the CD-ROM

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

- If you have a SunOS 4.1.3, SunOS 5.0, or SunOS 5.1, as root, enter:

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

- If you have SunOS 5.2 (Solaris 2.2) or higher, enter:

```
% cd ↵
% eject ↵
```

3.3 Setting Up the Environment

Before or after you install the tools, you can define *environment variables* that set certain code generation tool parameters you normally use. An environment variable is a system symbol that you define and assign to a string. When you use environment variables, default values are set, making each individual invocation of the tools simpler because these parameters are automatically specified. When you invoke a tool, you can use command-line options to override many of the defaults that are set with environment variables.

To set up the environment, enter these commands. Be sure you are not logged on as root.

- For C shells:

  setenv C_DIR "*tool_dir***"** ⏎
  setenv A_DIR "*tool_dir***"** ⏎
  set path=(*tool_dir* **$path)** ⏎

- For Bourne or Korn shells:

  C_DIR=*tool_dir* ⏎
  A_DIR=*tool_dir* ⏎
  PATH=*tool_dir***:$PATH** ⏎

You can move these commands into your .login or .cshrc file (for C shells) or .profile file (for Bourne or Korn shells) to avoid entering the commands each time you invoke a new shell.

The remainder of this section describes these environment variables and other variables that you can define.

Identifying the directory that contains the executable files (path statement)

You must include the *tool_dir* directory in your path statement. This allows you to specify the assembler and compiler tools without specifying the name of the directory that contains the executable files.

- If you modify your .cshrc file (for C shells) or .profile file (for Bourne or Korn shells) to change the path information, add the following to the end of the path statement:

  tool_dir

- If you set the path statement from the command line, use this format:

  - For C shells:

    set path=(*tool_dir* **$path)**

  - For Bourne or Korn shells:

    PATH=*tool_dir* **:$PATH**

  The addition of **$path**/**$PATH** ensures that this path statement does not undo the path statements in the .cshrc or .profile file.

Identifying alternate directories for the assembler (A_DIR)

By default, the assembler searches for copy/include files or macro libraries in the current directory and then in directories named by the –i (name alternate directories) option. Use the A_DIR environment variable to define additional search paths. The format of the command for assigning the environment variable is as follows:

- For C shells:

  setenv A_DIR ***"****pathname1*;*pathname2* … ***"***

- For Bourne or Korn shells:

  A_DIR=*"****pathname1*;*pathname2* … ***"***
  export A_DIR

You can separate the pathnames with a semicolon or a blank.

Once you set A_DIR, you can use the .copy, .include, or .mlib directive in assembly source without specifying path information.

For more information on the –i option, see the *TMS320C54x Assembly Language Tools User's Guide* or the *TMS320C54x Optimizing C Compiler User's Guide*.

Identifying alternate directories for the compiler (C_DIR)

By default, the compiler searches the current directory for #include files and object libraries such as the runtime-support and C I/O libraries. Use the C_DIR environment variable to define additional search paths. The format of the command for assigning the environment variable is as follows:

- For C shells:

  setenv C_DIR ***"****pathname1*;*pathname2* … ***"***

- For Bourne or Korn shells:

  C_DIR=*"****pathname1*;*pathname2* … ***"***
  export C_DIR

You can separate pathnames with a semicolon or with blanks.

Once you set C_DIR, you can use the #include directive in your C source code without specifying path information.

Setting default shell options (C_OPTION)

When using the shell program (cl500), you might find it useful to set default options for the compiler, assembler, and linker using the C_OPTION environment variable. After reading the command line options and the input filenames, the shell reads the contents of the C_OPTION environment variable. The options and/or input filenames that you define with C_OPTION are used every time you run the shell.

Setting up default options with the C_OPTION environment variable is especially useful when you want to run consecutive times with the same set of options and/or input files.

The syntax for the C_OPTION environment variable is:

- For C shells:

  setenv C_OPTION *"option1 option2 … "*

- For Bourne or Korn shells:

  C_OPTION=*"option1 option2 … "*
  export C_OPTION

Options specified in the environment variable are specified in the same way and have the same meaning as they do on the command line. For more information about compiler and assembler options, see the *TMS320C54x Optimizing C Compiler User's Guide* and the *TMS320C54x Assembly Language Tools User's Guide*.

For example, if you want to always run quietly (the –q option), enable C source interlisting (the –s option), and link (the –z option), set up the C_OPTION environment variable as follows:

```
setenv C_OPTION "-qs -z"     ; for C shells
C_OPTION="-qs -z"            ; for Bourne or Korn shells
export C_OPTION
```

In this example, each time you run the shell program, it runs the linker. Any options following –z on the command line or in C_OPTION are passed to the linker. This enables you to use the C_OPTION environment variable to specify default compiler and linker options and then specify additional compiler and linker options on the shell command line. If you have set –z in the environment variable and want to compile only, use the –c option of the shell. These additional examples assume C_OPTION is set as shown above:

```
cl500  *.c                ; compiles and links
cl500  -c *.c             ; only compiles
cl500  *.c -z lnk.cmd     ; compiles and links using a
cl500                     ; command file
cl500  -c *.c -z lnk.cmd  ; only compiles (-c overrides -z)
```

For more information on linker options, see the *Linker Options* section in the *TMS320C54x Assembly Language Tools User's Guide.*

Specifying a temporary file directory (TMP)

The shell program creates intermediate files as it processes your program. For example, the parser phase of the shell creates a temporary file used as input by the code generator phase. By default, the shell puts intermediate files in the current directory. However, you can name a specific directory for temporary files by using the TMP environment variable.

This feature allows use of a RAM disk or other file systems. It also allows source files to be compiled from a remote directory without writing any files into the directory where the source resides; this is useful for protected directories.

To set the TMP environment variable, use this syntax:

- For C shells:

  ```
  setenv TMP "/temp"
  ```

- For Bourne or Korn shells:

  ```
  TMP="/temp"
  export TMP
  ```

Resetting defined environment variables

The environment variables that you define remain set until you reboot the system. If you want to clear an environment variable, use this command:

- For C shells:

  unsetenv *variable name*

- For Bourne or Korn shells:

  unset *variable name*

For example, to reset the A_DIR environment variable, enter one of these commands:

- For C shells:

  ```
  unsetenv A_DIR
  ```

- For Bourne or Korn shells:

  ```
  unset A_DIR
  ```

3.4 Where to Go From Here

Your code generation tools are now installed. At this point, you should do the following:

- ❑ Go to Chapter 5, *Getting Started With the Code Generation Tools*. This chapter provides you with an overview of how to invoke and use the assembler, linker, and compiler.
- ❑ Read Chapter 6, *Release Notes*, to understand the new features included in the 1.20 release of the code generation tools.

Chapter 4

Setting Up the Tools on an HP Workstation

This chapter helps you install release 1.20 of the TMS320C54x code generation tools and set up your code-development environment on an HP 9000 Series 700™ computer with HP-UX™ 9.0 or higher. These tools include an optimizing C compiler and a full set of assembly language tools for developing and manipulating assembly language and object (executable) code.

The C compiler tools include:

- ❑ Parser
- ❑ Optimizer
- ❑ Code generator
- ❑ Interlist utility
- ❑ Library-build utility

The assembly language tools are composed of the following:

- ❑ Assembler
- ❑ Archiver
- ❑ Linker
- ❑ Absolute lister
- ❑ Cross-reference lister
- ❑ Hex conversion utility
- ❑ Mnemonic to algebraic assembly code translator

Topic **Page**

4.1 System Requirements 4-2
4.2 Mounting the CD-ROM and Installing the Tools 4-3
4.3 Setting Up the Environment 4-4
4.4 Where to Go From Here 4-8

4.1 System Requirements

To install the TMS320C54x code generation tools on an HP workstation, you need the following items:

- ❑ HP 9000 Series 700 computer
- ❑ 4 Mbytes of disk space for the software tools
- ❑ HP-UX 9.0 or higher operating system
- ❑ CD-ROM drive
- ❑ Root privileges to mount and unmount the CD-ROM

4.2 Mounting the CD-ROM and Installing the Tools

To install the software tools, you must mount the CD-ROM, copy the files to your system, and unmount the CD-ROM.

Mounting the CD-ROM

The code generation tools package is shipped on 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/hp700 ↵

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

Installing the tools

Be sure you are not logged on as root. To install the software tools, follow these steps:

1) If you don't already have a tools directory, create one. Enter:

   mkdir *tool_dir* ↵

   Replace *tool_dir* with your own directory name, including the path information, to install the tools.

2) Copy the files to your directory:

   cp –r * *tool_dir* ↵

Unmounting the CD-ROM

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

 # cd ↵
 # umount /cdrom ↵
 # exit ↵

4.3 Setting Up the Environment

Before or after you install the tools, you can define *environment variables* that set certain code generation tool parameters you normally use. An environment variable is a system symbol that you define and assign to a string. When you use environment variables, default values are set, making each individual invocation of the tools simpler because these parameters are automatically specified. When you invoke a tool, you can use command-line options to override many of the defaults that are set with environment variables.

To set up the environment, enter these commands. Be sure you are not logged on as root.

- For C shells:

  setenv C_DIR "*tool_dir***"** ↵
  setenv A_DIR "*tool_dir***"** ↵
  set path=(*tool_dir* **$path)** ↵

- For Bourne or Korn shells:

  C_DIR=*tool_dir* ↵
  A_DIR=*tool_dir* ↵
  PATH=*tool_dir***:$PATH** ↵

You can move these commands into your .login or .cshrc file (for C shells) or .profile file (for Bourne or Korn shells) to avoid entering the commands each time you invoke a new shell.

The remainder of this section describes these environment variables and other variables that you can define.

Identifying the directory that contains the executable files (path statement)

You must include the *tool_dir* directory in your path statement. This allows you to specify the assembler and compiler tools without specifying the name of the directory that contains the executable files.

- If you modify your .cshrc file (for C shells) or .profile file (for Bourne or Korn shells) to change the path information, add the following to the end of the path statement:

  tool_dir

- If you set the path statement from the command line, use this format:

  - For C shells:

    set path=(*tool_dir* **$path)**

  - For Bourne or Korn shells:

    PATH=*tool_dir*: **$PATH**

  The addition of **$path**/**$PATH** ensures that this path statement does not undo the path statements in the .cshrc or .profile file.

Identifying alternate directories for the assembler (A_DIR)

By default, the assembler searches for copy/include files or macro libraries in the current directory and then in directories named by the –i (name alternate directories) option. Use the A_DIR environment variable to define additional search paths. The format of the command for assigning the environment variable is as follows:

- For C shells:

  setenv A_DIR ***"****pathname1*;*pathname2 … "*

- For Bourne or Korn shells:

  A_DIR=*"pathname1*;*pathname2 … "*
  export A_DIR

You can separate the pathnames with a semicolon or a blank.

Once you set A_DIR, you can use the .copy, .include, or .mlib directive in assembly source without specifying path information.

For more information on the –i option, see the *TMS320C54x Assembly Language Tools User's Guide* or the *TMS320C54x Optimizing C Compiler User's Guide*.

Identifying alternate directories for the compiler (C_DIR)

By default, the compiler searches the current directory for #include files and object libraries such as the runtime-support and C I/O libraries. Use the C_DIR environment variable to define additional search paths. The format of the command for assigning the environment variable is as follows:

- For C shells:

  setenv C_DIR ***"****pathname1*;*pathname2 … "*

- For Bourne or Korn shells:

  C_DIR=*"pathname1*;*pathname2 … "*
  export C_DIR

You can separate pathnames with a semicolon or with blanks.

Once you set C_DIR, you can use the #include directive in your C source code without specifying path information.

Setting default shell options (C_OPTION)

When using the shell program (cl500), you might find it useful to set default options for the compiler, assembler, and linker using the C_OPTION environment variable. After reading the command line options and the input filenames, the shell reads the contents of the C_OPTION environment variable. The options and/or input filenames that you define with C_OPTION are used every time you run the shell.

Setting up default options with the C_OPTION environment variable is especially useful when you want to run consecutive times with the same set of options and/or input files.

The syntax for the C_OPTION environment variable is:

- For C shells:

  setenv C_OPTION ***"****option1 option2 … ****"***

- For Bourne or Korn shells:

  C_OPTION=*"option1 option2 … "*
  export C_OPTION

Options specified in the environment variable are specified in the same way and have the same meaning as they do on the command line. For more information about compiler and assembler options, see the *TMS320C54x Optimizing C Compiler User's Guide* and the *TMS320C54x Assembly Language Tools User's Guide*.

For example, if you want to always run quietly (the –q option), enable C source interlisting (the –s option), and link (the –z option), set up the C_OPTION environment variable as follows:

```
setenv C_OPTION "-qs -z"     ; for C shells
C_OPTION="-qs -z"            ; for Bourne or Korn shells
export C_OPTION
```

In this example, each time you run the shell program, it runs the linker. Any options following –z on the command line or in C_OPTION are passed to the linker. This enables you to use the C_OPTION environment variable to specify default compiler and linker options and then specify additional compiler and linker options on the shell command line. If you have set –z in the environment variable and want to compile only, use the –c option of the shell. These additional examples assume C_OPTION is set as shown above:

```
cl500  *.c               ; compiles and links
cl500  -c *.c            ; only compiles
cl500  *.c -z lnk.cmd    ; compiles and links using a
cl500                    ; command file
cl500  -c *.c -z lnk.cmd ; only compiles (-c overrides -z)
```

For more information on linker options, see the *Linker Options* section in the *TMS320C54x Assembly Language Tools User's Guide.*

Specifying a temporary file directory (TMP)

The shell program creates intermediate files as it processes your program. For example, the parser phase of the shell creates a temporary file used as input by the code generator phase. By default, the shell puts intermediate files in the current directory. However, you can name a specific directory for temporary files by using the TMP environment variable.

This feature allows use of a RAM disk or other file systems. It also allows source files to be compiled from a remote directory without writing any files into the directory where the source resides; this is useful for protected directories.

To set the TMP environment variable, use this syntax:

- For C shells:

  ```
  setenv TMP "/temp"
  ```

- For Bourne or Korn shells:

  ```
  TMP="/temp"
  export TMP
  ```

Resetting defined environment variables

The environment variables that you define remain set until you reboot the system. If you want to clear an environment variable, use this command:

- For C shells:

  unsetenv *variable name*

- For Bourne or Korn shells:

  unset *variable name*

For example, to reset the A_DIR environment variable, enter one of these commands:

- For C shells:

  ```
  unsetenv A_DIR
  ```

- For Bourne or Korn shells:

  ```
  unset A_DIR
  ```

4.4 Where to Go From Here

Your code generation tools are now installed. At this point, you should do the following:

- ❑ Go to Chapter 5, *Getting Started With the Code Generation Tools*. This chapter provides you with an overview of how to invoke and use the assembler, linker, and compiler.
- ❑ Read Chapter 6, *Release Notes*, to understand the new features included in the 1.20 release of the code generation tools.

Chapter 5

Getting Started With the Code Generation Tools

This chapter helps you start using the assembler, linker, and compiler tools by providing basic startup information. For more information about invoking and using these tools, see the *TMS320C54x Assembly Language Tools User's Guide* and the *TMS320C54x Optimizing C Compiler User's Guide*.

Topic **Page**

5.1 Getting Started With the Assembler and Linker 5-2

5.2 Getting Started With the C Compiler 5-6

5.1 Getting Started With the Assembler and Linker

This section provides a quick walkthrough of the assembler and linker so that you can get started without reading the entire *TMS320C54x Assembly Language Tools User's Guide*. These examples show the most common methods for invoking the assembler and linker.

Create two short source files to use for the walkthrough; call them file1.asm and file2.asm. (See Example 5–1 and Example 5–2.)

Example 5–1. file1.asm

```
            .global TEST

START   RESET

LOOP    CALL TEST
        BC LOOP,AGEQ
        .end
```

Example 5–2. file2.asm

```
            .global TEST

TEST    MAR  *AR3+
        BC   SKP, ALEQ
        MAR  *AR2+

SKP     ADD  *AR3, A
        B    TEST
        .end
```

1) Enter the following command to assemble file1.asm:

   asm500 file1 ↵

   The asm500 command invokes the assembler. The input source file is file1.asm. (If the input file extension is .asm, you don't have to specify the extension; the assembler uses .asm as the default.)

   This example creates an object file called file1.obj. The assembler creates an object file only if there are no errors. You can specify a name for the object file, but if you don't, the assembler uses the input filename with an extension of .obj.

2) Now enter the following command to assemble file2.asm:

   asm500 file2.asm –l ↵

   This time, the assembler creates an object file called file2.obj. The –l (lowercase L) option tells the assembler to create a listing file; the listing file for this example is called file2.lst. It is not necessary to create a listing file, but it gives you information and assures you that the assembly has resulted in the desired object code. The listing file for this example is shown in Example 5–3.

Example 5–3. file2.lst, the Listing File Created by asm500 file2.asm –l

```
TMS320C54x COFF Assembler   Version 1.20         Wed Oct  9 11:06:11 1997
(c) Copyright 1997  Texas Instruments Incorporated
asm500 file2.asm –l

file2.asm                                                          PAGE    1

          1                    .global TEST
          2 000000 6D93    TEST   MAR    *AR3+
          3 000001  F847          BC     SKP, ALEQ
            000002  0004'
          4 000003  6D92          MAR    *AR2+
          5
          6 000004 0083    SKP    ADD    *AR3, A
          7 000005 F073           B      TEST
            000006 0000'
          8                      .end

 No Errors,  No Warnings
```

3) Now enter the following command to link file1.obj and file2.obj:

   lnk500 file1 file2 –m lnker2.map –o prog.out ↵

The lnk500 command invokes the linker. The input object files are file1.obj and file2.obj. (If the input file extension is .obj, you don't have to specify the extension; the linker uses .obj as the default.) The linker combines file1.obj and file2.obj to create an executable object module called prog.out. The –o option supplies the name of the output module. Example 5–4 shows the map file resulting from this operation. The map file is produced only if you use the –m option.

Example 5–4. Output Map File, lnker2.map

```
**************************************************
TMS320C54x COFF Linker        Version 1.20
**************************************************
Wed Oct  9 11:14:32 1997
OUTPUT FILE NAME:   <prog.out>
ENTRY POINT SYMBOL: 0

MEMORY CONFIGURATION

          name     origin    length      used    attributes    fill
         --------  --------  ---------  --------  ----------  --------
   PAGE 0: PROG     00000080  00000ff00  0000000c     RWIX
   PAGE 1: DATA     00000080  00000ff80  00000000     RW

SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
.text      0    00000080    0000000c
                  00000080    00000000     file2.obj (.text)
                  00000085    00000000     file1.obj (.text)

.data      0    00000000    00000016     UNINITIALIZED
                  00000000    00000000     file1.obj (.data)
                  00000000    00000000     file2.obj (.data)

.bss       1    00000000    00000000     UNINITIALIZED
                  00000000    00000000     file2.obj (.bss)
                  00000000    00000000     file1.obj (.bss)

GLOBAL SYMBOLS

address  name                            address  name
-------- ----                            -------- ----
00000000 .bss                            00000000 .data
00000000 .data                           00000000 edata
00000080 .text                           00000000 .bss
00000085 TEST                            00000000 end
00000000 __lflags                        00000000 __lflags
00000000 edata                           00000080 .text
00000000 end                             00000085 TEST
0000008c etext                           0000008c etext

[8 symbols]
```

5.2 Getting Started With the C Compiler

The TMS320C54x C compiler consists of two passes: the first pass parses the code, and the second pass produces a single assembly language source file that must be assembled and linked. You can use an optional optimization pass after the first pass. The simplest way to compile, assemble, and link a C program is to use the compiler shell program with the –z option. This section provides a quick walkthrough so that you can get started without reading the entire *TMS320C54x Optimizing C Compiler User's Guide*.

1) Create a sample file called function.c that contains the following code:

```
/************************************/
/*          function.c              */
/* (Sample file for walkthrough)    */
/************************************/
int main(i)
   int i;
{
   return(i < 0 ? –i : i );
}
```

2) To invoke the shell program to compile and assemble function.c, enter:

`cl500 –o function` ⏎

The –o option invokes the optimizer at the default level. The shell program prints the following information as it compiles the program:

```
[function]
TMS320C54x ANSI C Compiler   Version 1.20
Copyright (c) 1997     Texas Instruments Incorporated
   ″function.c″   ==> main
TMS320C54x ANSI C Optimizer   Version 1.20
Copyright (c) 1997     Texas Instruments Incorporated
   ″function.c″   ==> main
TMS320C54x ANSI C Codegen    Version 1.20
Copyright (c) 1997     Texas Instruments Incorporated
   ″function.c″:  ==> main
TMS320C54x COFF Assembler   Version 1.20
Copyright (c) 1997    Texas Instruments Incorporated
 PASS 1
 PASS 2

 No Errors,  No Warnings
```

The shell program runs the two compiler passes, the optimizer, and the assembler as follows:

ac500 → C parser
opt500 → Optimizer
cg500 → Code generator
asm500→ Assembler

By default, the shell deletes the assembly language file from the current directory after the file is assembled. If you want to inspect the assembly language output, use the –k option to retain the assembly language file:

```
cl500 –o –k function
```

3) Also by default, the shell creates a COFF object file as output; however, if you use the –z option, the output is an *executable* object module. The following examples show two ways of creating an executable object module:

   a) The example in step 2 creates an object file called function.obj. To create an executable object module, run the linker separately by invoking lnk500 as in the following example:

      ```
      lnk500 –c function.obj lnk.cmd –o function.out –l rts.lib
      ```

      The –c linker option tells the linker to use the ROM autoinitialization model. The linker command file, lnk.cmd, is shipped with the code generation tools. The –o option names the output module, function.out; if you don't use the –o option, the linker names the output module a.out. The –l option names the runtime-support library. You must have a runtime-support library before you can create an executable object module; the prebuilt runtime-support library, rts.lib, is included with the code generation tools.

   b) In this example, use the –z option, which tells the shell program to run the linker. The –z option is followed by linker options.

      ```
      cl500 –o function.c –z lnk.cmd –o function.out –l rts.lib
      ```

      This example runs the two compiler passes, the optimizer, the assembler, and the linker as follows:

      ac500 → C parser
      opt500 → Optimizer
      cg500 → Code generator
      asm500→ Assembler
      lnk500 → Linker

For more information on linker commands, see the *Linker Description* chapter of the *TMS320C54x Assembly Language Tools User's Guide* and the *Linking C Code* chapter of the *TMS320C54x Optimizing C Compiler User's Guide.*

4) The TMS320C54x compiler package also includes an *interlist utility*. This program interlists the C source statements as comments in the assembly language compiler output, allowing you to inspect the assembly language generated for each line of C. To run the interlist utility, invoke the shell program with the –ss option. For example:

```
cl500 –ss function –z lnk.cmd –o function.out
```

The output of the interlist utility is written to the assembly language file created by the compiler. (The shell –ss option implies –k; that is, when you use the interlist utility, the assembly file is automatically retained.)

Chapter 6

Release Notes

This chapter contains documentation of tools and features that are new or have been changed since the last release.

Topic **Page**

6.1 ’C54x Silicon Problem Workaround 6-2
6.2 Release Enhancements 6-3
6.3 Fixed Problems 6-6
6.4 Undocumented Features 6-10

6.1 'C54x Silicon Problem Workaround

A compiler option (cl500 option –mx, cg500 option –x) allows you to avoid first run 'C54x silicon bugs. Use of this switch may be necessary when preparing a program for use with 'C54x silicon device versions earlier than 2.2.

The –mx option detects an instruction sequence with a write to memory followed by a dual read from memory and a long read from memory. To protect against processor lockups, the compiler inserts an NOP between the read instructions.

6.2 Release Enhancements

This section lists the release enhancements for version 1.20 of the TMS320C54x code generation tools.

Assembly Language Tools

See the *TMS320C54x Assembly Language Tools User's Guide* chapter reference given with each bullet for details.

- The assembler produces COFF2 format. COFF2 differs from COFF1 format (produced in previous releases) in the following ways:
  - The size of section names is no longer limited to 8 characters.
  - The size of file names is no longer limited to 14 characters.

  See Appendix A for more information.

- The assembler now accepts algebraic assembly. Algebraic assembly is turned on when the assembler is invoked with the –mg option, or when the first line of an assembly source file contains a .algebraic directive as the first line of the source file. See Chapter 6 for more information.

  Note that algebraic and mnemonic assembly syntax cannot be mixed within the same assembly source file.

- A mnemonic–to–algebraic translator is included. See Chapter 13 for more information.

- The linker produces COFF2 format by default. The COFF format produced by the linker may be selected with the –v linker option:

–v0	causes linker to produce COFF version 0 format
–v1	causes linker to produce COFF version 1 format
–v2	causes linker to produce COFF version 2 format (default)

  There should be no reason to have the linker produce COFF0; the –v0 option is included for completeness. See Chapter 9 for more information.

- The hex conversion utility has been updated to create boot load tables for the C548, C545LP, and C546LP devices. It supports the following boot load options for these devices:

–bootorg WARM	Specify the source of the boot loader table as the table currently in memory
–bootorg SERIAL	Specify the source of the boot loader table as the serial port
–bootorg PARALLEL	Specify the source of the boot loader table as the parallel port
–bootorg COMM	Specify the source of the boot loader table as the communications port
–spc *value*	Set the serial port control register value
–spce *value*	Set the serial port control extension register value
–trta *value*	Set the TDM serial port receive/transmit address register value
–tcsr *value*	Set the TDM serial port channel select register value
–swwsr *value*	Set the Software Wait State Reg value for PARALLEL/WARM boot mode
–bscr *value*	Set the Bank-Switch Control Reg value for PARALLEL/WARM boot mode

  See Chapter 12 for more information.

- The hex conversion utility includes the –m1, –m2, and –m3 options to support the Motorola S1, S2, and S3 formats, respectively. See Chapter 12 for more information.

C compiler notes

See the *TMS320C54x Optimizing C Compiler User's Guide* chapter reference given with each bullet for details.

- The compiler shell now accepts the following options:

–@ *filename*	accepts a command file containing compiler options as input. This facilitates compilation when the command line limit has been exceeded.
–ga	writes a .global statement for all extern variables
–mf	interprets all call instructions as far calls, and interprets all return instructions as far returns. A far call calls a function outside the 16-bit range, and a far return returns a value from outside the 16-bit range.
–v*value*	determines which processor instructions are built for. Use one of the following for *value*: 541 542 543 545 545lp 546lp 548

  See Chapter 2 for more information.

- The compiler supports the following pragmas:

  #pragma CODE_SECTION(*symbol***, "***section name***");**

  The CODE_SECTION pragma allows code for the specified symbol to be assembled into the named section. This pragma facilitates linking code for specific functions into various memory locations, which can be particularly useful when implementing applications that use the extended address space.

  #pragma DATA_SECTION(*symbol***, "***section name***");**

  The DATA_SECTION pragma allows space to be reserved for a symbol in a section other than .bss. This pragma facilitates linking applications using the extended address space.

  See Chapter 4 for more information on these pragmas.

6.3 Fixed Problems

This section lists the known problems that have been fixed in version 1.20 of the TMS320C54x code generation tools. A complete list of fixed problems is provided in the README file included with the tools at installation.

Assembly Language Tools

SDSsq02266 When a two word instruction with a long constant as the second word of the encoded instruction occurs on a 2048 word boundary, the assembler may assign incorrect relocation information on this constant. The incorrect relocation information will cause the linker to fail to properly relocate the constant. The constant will have the value assigned to it by the section program counter (SPC) within the assembler.

SDSsq02531 The assembler does not treat signed floating-point constant operations correctly in all instances. The example below:

```
t2 .set –5.0
...
t15 .set t2 * –1.0
```

incorrectly yields a negative value.

SDSsq02816 There is a bug in the parser of the assembler for STH instructions in which it fails to catch incorrect operand combinations such as:

```
STH A, *AR1+,1
```

The side effect of this is that the assembler will generate errors when processing addresses of labels that subsequently appear in the file.

SDSsq02923 The assembler may incorrectly parse an assembly line when it contains a local macro label. In the example below:

```
mymac .macro
       bd    loc?
       nop
       nop
loc?  .endm
```

the last line may be incorrectly parsed.

SDSsq03078 The assembler may issue an erroneous error message when any expression follows an expression involving load labels. In the code below, the problem occurs when evaluating the forced substitution symbol.

```
ADDX .macro ABC
     .var  TMP
     .asg :ABC(1):,TMP
     .if $syncmp(TMP, “#”) = 0
     ADD  ABC, A
     .else
     .emsg “Bad Macro Parameter”
     .endif
     .endm RPT #(LABEL2 – LABEL1)
      NOP

      ADDX #100
```

An error message occurs when processing the ABC(1) expression in the macro expansion.

SDSsq03214 The linker may core dump when a linker-defined fill section is created during the link.

SDSsq03349 The hex conversion utility may fail with a Segmentation Fault under some circumstances.

SDSsq03400 The hex conversion utility incorrectly handles the case where a section may span more than one fileset. For example:

```
ROMS { PAGE 0 : ROM0 : origin = 0x0000
                       length = 0x1000
                       files = { rom0 }
                ROM1 : origin = 0x1000
                       length = 0x1000
                       files = { rom1 } }
```

If a section in the input COFF file begins at address 0x800 and has a length of 0x800 hex, a portion of the raw data from the output section will be written to the file “rom0” and the remainder to the file “rom1”.

SDSsq03409 The linker may abort abnormally when an output section is formed from input sections that do not have the same name as the output section. For example:

```
SECTIONS { .txt : { *(input) }
```

may cause an abnormal termination of the link.

C Compiler

SDSsq02116 The code generator may generate internal error message “...COMPILER ERROR: no match for VREG” when generating code for interrupt service routines.

SDSsq02683 (PC only) If the command line (including options given on the command line plus any specified by C_OPTION) exceeds 99 characters, the compiler may fail, generating the error message: “COMPILER ERROR:PACKET ERROR:8”

SDSsq02790 The compiler may generate incorrect code in which an invalid register name is given.

SDSsq02791 When –g is used, the compiler may inadvertently misplace a SSXM instruction that is needed to load 16-bit values into the accumulator prior to performing an ABS. This may result in an incorrect answer being returned for an ABS of a 16-bit value.

SDSsq02813 The RTS C boot routine c_int0 does not contain code for the copy of the .const section from program to data space as is stated in the *TMS320C54x Optimizing C Compiler User’s Guide* (SPRU103A, Section 4.1.3, pg. 4–4).

SDSsq02859 The compiler may err when generating code for an assignment from the read of a PORT variable, when the target of the assignment is a local var whose storage space is allocated on the stack, and whose value is subsequently used in a call to a function. For example:

```
port unsigned int port100;
func()
{
 int tmp;
 tmp = port100;
 call(0,tmp);
}
```

An incorrect address may be calculated for the store to tmp.

SDSsq03086 A shift instruction within the 1&&MPY instruction does not fully sign extend the result. This may cause incorrect values for the result when a subsequent operation occurs before the result is stored to memory.

SDSsq03103 The compiler generates incorrect code for a symbol declared as an ioport and used in a conditional expression.

SDSsq03105 The compiler may generate the internal error message “no match for NAME” when processing an expression involving a symbol declared as an ioport in which the compiler must generate code to read the value of the port into a memory-mapped accumulator.

SDSsq03303 The compiler fails to properly align a long constant on a double word boundary. For example, in

```
const long A = 0x2ffff;
```

symbol A may not be properly aligned.

SDSsq03310 Same as 03103.

6.4 Undocumented Features

This section lists features that were undocumented in the 1997 *TMS320C54x Assembly Language Tools User's Guide*.

Assembler

–g	causes the assembler to generate symbolic debug directives for use with the source-level debuggers
–v*value*	determines the processor for which instructions are built. Use one of the following for *value*: 541 542 543 545 545lp 546lp 548

Hex Conversion Utility

–warm	Specify the source of the boot loader table as the table currently in memory

Appendix A

Troubleshooting DOS Systems

DOS/4GW is a memory manager that is embedded into the TMS320C54x code generation tools, so you may occasionally see DOS/4GW error messages while you are using the tools. The executable files for DOS/4GW are not shipped as such, nor is any documentation provided on this tool, except for the list of error messages.

Section A.2, *Kernel Error Messages*, and Section A.3, *DOS/4G Error Messages*, are excerpted from the *DOS/4GW User's Manual* (reproduced here with the permission of Tenberry Software, Inc.). Included are lists of error messages with descriptions of the circumstances in which the error is most likely to occur and suggestions for remedying the problem. (Portions of the excerpt have been modified to provide you with specific information about using TI tools.)

Topic **Page**

A.1 Troubleshooting in the Protected-Mode Environment A-2

A.2 Kernel Error Messages A-5

A.3 DOS/4G Error Messages A-9

A.1 Troubleshooting in the Protected-Mode Environment

Getting 32-bit programs to execute properly under DOS can be frustrating. Your computer's configuration and memory management can cause problems that may be difficult to find because many programs are interacting.

This list of error messages is reproduced here because they may occur when executing any tools, since all of the tools have been assembled along with the DOS/4GW memory extender. When reading this material, keep these considerations in mind:

- When an *Action* directs you to technical support, determine the configuration of your system by using the **PMINFO** (on page A-3) programs before contacting technical support:

  To contact technical support, call the following telephone number:

  Microcontroller Hotline (713) 274–2370

- Some error messages are not included in this section because they are rarely seen when using DOS/4GW with the TMS320C54x tools. Also, many of the messages that *are* documented here are seldom seen when using DOS/4GW with the TMS320C54x tools. Nevertheless, you may find this text to be useful in debugging your programs.

Should you encounter any error message not listed here, or should problems persist, contact technical support as directed above.

The PMINFO.EXE program

Purpose: Run PMINFO.EXE to determine the performance of protected/real-mode switching and extended memory.

Notes: The time-based measurements made by PMINFO may vary slightly from run to run.

If this error message appears:

DOS/16M error: [17] system software does not follow VCPI or DPMI specifications

check for a statement in your CONFIG.SYS containing **NOEMS**. If such a statement exists, remove it and reboot your computer.

If the computer is not equipped with extended memory or if none is available for DOS/4GW, the extended-memory measurements will not display.

Other DOS/4GW error messages are in Section A.3, *DOS/4G Error Messages*.

Example: The following example shows the output of the PMINFO program on an 80486 AT-compatible machine running at 33 MHz.

```
-================================ PMINFO ======================================-

          Protected Mode and Extended Memory Performance Measurement -- 4.45
                  Copyright (c) Tenberry Software, Inc. 1987 - 1995

      DOS memory   Extended memory   CPU performance equivalent to 33.0 MHz 80486
      ----------   ---------------
             640             17854   K bytes configured (according to BIOS).
             640             31744   K bytes physically present (SETUP).
             550             17585   K bytes available for DOS/16M programs.
      21.6 (0.0)        19.1 (0.5)   MB/sec word transfer rate (wait states).
      35.4 (0.5)        34.4 (0.5)   MB/sec 32-bit transfer rate (wait states).

      Overall cpu and memory performance (non-floating point) for typical
      DOS programs is 7.78 ± 0.62 times an 8MHz IBM PC/AT.

      Protected/Real switch rate = 18078/sec (55 µsec/switch, 33 up + 21 down),
      DOS/16M switch mode 11 (VCPI).
```

PMINFO provides the information shown in Table A–1.

Table A–1. PMINFO Fields

Measurement	Purpose
CPU performance	Shows the CPU processor equivalent and the speed of the CPU (in MHz).
According to BIOS	Shows the configured memory in DOS and extended memory as provided by the BIOS (interrupts 12h and 15h, function 88h).
SETUP	Shows the configuration obtained directly from the CMOS RAM as set by the computer's setup program. It is displayed only if the numbers are different from those in the BIOS line. They are different if the BIOS has reserved memory for itself or if another program has allocated memory and is intercepting the BIOS configuration requests to report less memory available than is physically configured.
DOS/16M programs	If displayed, shows the low and high addresses available to DOS/4GW in extended memory.
Transfer rates	PMINFO tries to determine the memory architecture. Some architectures perform well under some circumstances and poorly under others; PMINFO shows the best and worst cases. The architectures detected are cache, interleaved, page-mode (or static column), and direct. Measurements are made by using 32-bit accesses and are reported as the number of megabytes per second that can be transferred. The number of wait states is reported in parentheses. The wait states can be a fractional number, like 0.5, if there is a wait state on writes but not on reads. Memory bandwidth (that is, how fast the CPU can access memory) accounts for 60% to 70% of the performance for typical programs (those that are not heavily dependent on floating-point math).
Overall CPU and memory performance	Shows a performance metric developed by Tenberry Software, Inc. (formerly known as Rational Systems, Inc.), indicating the expected throughput for the computer relative to a standard 8-MHz IBM PC/AT (disk accesses and floating-point operations are both excluded).
Protected/real switch rate	Shows the speed with which the computer can switch between real and protected modes, both as the maximum number of round-trip switches that can occur per second, and as the time for a single round-trip switch, broken into the real-to-protected (up) and protected-to-real (down) components.

A.2 Kernel Error Messages

This section describes error messages from the DOS/16M kernel embedded in the TMS320C54x code generation tools. Kernel error messages can occur because of severe resource shortages, corruption of the executable file, corruption of memory, operating system incompatibilities, or internal errors. All of these messages are quite rare.

DOS/16M protected mode available only with 386 or 486

Description DOS/4G did not detect the presence of a 386-, 486-, or Pentium-based processor. You may see this error message even if you are using a 386 PC or later.

Action If you are running the tools on a 386 (or later) PC, rerun the program. If you are running the tools on a 286 PC, reinstall and run the tools on a 386 PC or later.

0: involuntary switch to real mode

Description The computer was in protected mode but switched to real mode without going through DOS/16M. This error most often occurs because of an unrecoverable stack segment exception (stack overflow) but can also occur if the Global Descriptor Table or Interrupt Descriptor Table is corrupted.

Action Restart your computer. If the problem persists, contact technical support.

2: not a DOS/16M executable <filename>

Description DOS4G.EXE or a bound DOS/4G application has probably been corrupted in some way.

Action Recopy the file from the source media.

6: not enough memory to load program

Description There is not enough memory to load DOS/4G.

Action Make more memory available and try again.

8: cannot open file <filename>

Description The DOS/16M loader cannot load DOS/4G, probably because DOS has run out of file units.

Action Set a larger FILES= entry in the CONFIG.SYS file, reboot, and try again.

9: cannot allocate tstack

Description There is not enough memory to load DOS/4G.

Action Make more memory available and try again.

10: cannot allocate memory for GDT

Description There is not enough memory to load DOS/4G.

Action Make more memory available and try again.

11: no passup stack selectors – GDT too small

Description There is an internal error in DOS/4G or an incompatibility with other software.

Action Contact technical support.

12: no control program selectors – GDT too small

Description There is an internal error in DOS/4G or an incompatibility with other software.

Action Contact technical support.

13: cannot allocate transfer buffer

Description There is not enough memory to load DOS/4G.

Action Make more memory available and try again.

14: premature EOF

Description DOS4G.EXE or a bound DOS/4G application has probably been corrupted.

Action Recopy the file from the source media.

15: protected mode available only with 386 or 486

Description DOS/4G requires an 80386 (or later) CPU. It cannot run on an 80286 (or earlier) CPU.

Action Reinstall and run the tools on a 386 (or later) PC.

17: system software does not follow VCPI or DPMI specifications

Description Some memory-resident program has put your 386 or 486 CPU into Virtual 8086 mode. This is done to provide special memory services to DOS programs, such as EMS simulation (EMS interface without EMS hardware) or high memory. In this mode, it is not possible to switch into protected mode unless the resident software follows a standard that DOS/16M supports (DPMI, VCPI, and XMS are the most common).

Action Contact the vendor of your memory-management software.

22: cannot free memory

Description Memory was probably corrupted during execution of your program.

Action Make more memory available and try again.

23: no memory for VCPI page table

Description There is not enough memory to load DOS/4G.

Action Make more memory available and try again.

24: VCPI page table address incorrect

Description This is an internal error.

Action Contact technical support.

25: cannot initialize VCPI

Description An incompatibility with other software was detected. DOS/16M has detected that VCPI is present, but VCPI returns an error when DOS/16M tries to initialize the interface.

Action Find the other software that uses VCPI and disable it (stop its execution).

28: memory error, avail loop

Description Memory was probably corrupted during execution of your program. Using an invalid or stale alias selector may cause this error. Incorrect manipulation of segment descriptors may also cause it.

Action Rerun the program and/or restart your computer.

29: memory error, out of range

Description Memory was probably corrupted during execution of your program. Writing through an invalid or stale alias selector may cause this error.

Action Check your source code for references to variables that are not declared or are no longer in scope.

32: DPMI host error (possibly insufficient memory)
33: DPMI host error (need 64K XMS)
34: DPMI host error (cannot lock stack)

Description Memory under DPMI is probably insufficient.

Action Under Windows, make more physical memory available by eliminating or reducing any RAM drives or disk caches. You can also edit DEFAULT.PIF so that at least 64K bytes of XMS memory is available to non-Windows programs. Under OS/2, increase the DPMI_MEMORY_LIMIT in the DOS box settings.

35: general protection fault

Description An internal error in DOS/4G was probably detected. Faults generated by your program should cause error 2001 instead.

Action Contact technical support.

38: cannot use extended memory: HIMEM.SYS not version 2

Description An incompatibility with an old version of HIMEM.SYS was detected.

Action Upgrade to a more recent copy of DOS or upgrade your DOS memory extender.

40: not enough available extended memory (XMIN)

Description An incompatibility with your memory manager or its configuration was detected.

Action Configure the memory manager to provide more extended memory or change memory managers.

A.3 DOS/4G Error Messages

DOS/4G errors are more common than kernel errors when using DOS/4G or DOS/4GW with the TMS320C54x code generation tools. They are usually related to an unknown path name, corrupt files, or memory problems. Memory problems can include inadequate memory, poor configuration, or corrupted memory.

1000 ”can’t hook interrupts”

Description A DPMI host has prevented DOS/4G from loading.

Action Contact technical support.

1001 ”error in interrupt chain”

Description A DOS/4G internal error was detected.

Action Contact technical support.

1003 ”can’t lock extender kernel in memory”

Description DOS/4G couldn’t lock the kernel in physical memory, probably because of a memory shortage.

Action Free some memory for the DOS/4G application.

1005 ”not enough memory for dispatcher data”

Description There is not enough memory for DOS/4G to manage user-installed interrupt handlers properly.

Action Free some memory for the DOS/4G application.

1007 ”can’t find file <program> to load”

Description DOS/4G could not open the specified program. The file probably does not exist. It is possible that DOS ran out of file handles or that a network or similar utility has prohibited read access to the program.

Action Make sure that the filename was spelled correctly.

1008 ”can’t load executable format for file <filename> [<error code>]”

Description DOS/4G did not recognize the specified file as a valid executable file. DOS/4G can load linear executables (LE and LX) and EXPs (BW).

Action Recopy the file from the source media.

3301 ”unhandled EMPTYFWD, GATE16, or unknown relocation”
3302 ”unhandled ALIAS16 reference to unaliased object”
3304 ”unhandled or unknown relocation”

Description If your program was built for another platform that supports the LINEXE format, it may contain a construct that DOS/4G does not currently support, such as a call gate. One of these messages may also appear if your program has a problem mixing 16- and 32-bit code. A linker error is another likely cause.

Action Check for viruses and reinstall the tools from the source media. If the problem persists, contact technical support.

Appendix B

Glossary

A

ANSI: American National Standards Institute. An organization that establishes standards voluntarily followed by industries.

asm500: The name of the command that invokes the assembler for the TMS320C54x.

assembler: A software program that creates a machine-language program from a source file that contains assembly language instructions, directives, and macro directives. The assembler substitutes absolute operation codes for symbolic operation codes, and absolute or relocatable addresses for symbolic addresses.

B

.bss: One of the default COFF sections. You can use the .bss directive to reserve a specified amount of space in the memory map that can later be used for storing data. The .bss section is uninitialized.

C

C compiler: A program that translates C source statements into assembly language source statements.

cl500: The name of the compiler shell program for the TMS320C54x. (Note that the second character in the shell name is a lowercase L.)

common object file format (COFF): A binary object file format that promotes modular programming by supporting the concept of *sections.*

D

DOS/4G: The base version for DOS/4GW. You may occasionally see this term in an error message. If so, refer to Appendix A, *Troubleshooting DOS Systems*, for the appropriate action.

DOS/4GW: A memory extender that is bound with the MS-DOS version of the TMS320C54x tools. *The executable DOS/4GW file is not shipped separately but is embedded within the other executables*. Error messages from DOS/4GW are included in Appendix A, *Troubleshooting DOS Systems*, to assist you in debugging. If you receive one of these error messages, contact technical support for assistance, and remember that the tools are shipped as object files with the memory extender embedded.

DOS/16M: The executable filename for a tool that is embedded in the TMS320C54x code generation tools. You may occasionally see this term in an error message. If so, refer to Appendix A, *Troubleshooting DOS Systems*, for the appropriate action.

E

environment variables: System symbols that you define and assign to a string. They are usually included in batch files (for example, in the AUTOEXEC.BAT file).

G

global: A kind of symbol that is either 1) defined in the current module and accessed in another, or 2) accessed in the current module but defined in another.

I

initialized section: A COFF section that contains executable code or initialized data. An initialized section can be built up with the .data, .text, or .sect directive.

interlist utility: A utility that inserts as comments your original C source statements into the assembly language output from the assembler. The C statements are inserted next to the equivalent assembly instructions.

L

linker: A software tool that combines object files to form an object module that can be allocated into TMS320C54x system memory and executed by the device.

listing file: An output file created by the assembler that lists source statements, their line numbers, and their effects on the section program counter (SPC).

lnk500: The name of the command that invokes the linker for the TMS320C54x.

M

map file: An output file, created by the linker, that shows the memory configuration, section composition, section allocation, and symbol definitions and the addresses at which the symbols were defined for your program.

O

optimization: Improvement in the execution speed of a program or in the reduction of the size of C programs.

P

pragma: Preprocessor directive that provides directions to the compiler about how to treat a particular statement.

protected-mode programs: 32-bit extended MS-DOS programs. These programs require an extended memory manager and run on 80386-, 80486-, and Pentium-based PCs only. Protected-mode programs can use all available RAM on the computer up to 64 Mbytes.

R

real mode: 16-bit native MS-DOS mode. This mode limits the available memory to 640K bytes. Calls to DOS may involve switching from protected to real mode. *DOS real-mode tools are no longer supported by the TMS320C54x code generation tools.*

S

section: A relocatable block of code or data that will ultimately occupy contiguous space in the TMS320C54x memory map.

static variable: A kind of variable whose scope is confined to a function or a program. The values of static variables are not discarded when the function or program is exited; their previous value is resumed when the function or program is re-entered.

string table: A table that stores symbol names that are longer than eight characters (symbol names of eight characters or longer cannot be stored in the symbol table; instead, they are stored in the string table). The name portion of the symbol's entry points to the location of the string in the string table.

subsection: A relocatable block of code or data that will ultimately occupy contiguous space in the TMS320C54x memory map. Subsections are smaller sections within larger sections. Subsections give you tighter control of the memory map.

swap file: The file where virtual memory (secondary memory) is allocated on the hard disk.

symbolic debugging: The ability of a software tool to retain symbolic information so that it can be used by a debugging tool such as a simulator or an emulator.

T

.text: One of the default COFF sections. The .text section is an initialized section that contains executable code. You can use the .text directive to assemble code into the .text section.

U

uninitialized section: A COFF section that reserves space in the memory map but that has no actual contents. These sections are built up with the .bss and .usect directives.

V

virtual memory: The ability of a program to use more memory than a computer actually has available as RAM. This is accomplished by using a swap file on disk to augment RAM. When RAM is not sufficient, part of the program is swapped out to a disk file until it is needed again. The combination of the swap file and available RAM is the virtual memory. The TMS320C54x tools use the DOS/4GW memory extender to provide virtual-memory management (VMM). This memory extender is not provided as an executable file but is embedded in several of the tools shipped by TI. Contact technical support for more information.

Index

A

A_DIR environment variable
  for DOS 1-5
  for HP workstations 4-5
  for OS/2 2-5
  for SPARCstations 3-6
  for Windows 3.x 1-5
ANSI, defined B-1
asm500
  defined B-1
  invoking 5-3
assembler, defined B-1
assembler walkthrough 5-2 to 5-4

B

.bss section, defined B-1

C

C compiler, defined B-1
C compiler walkthrough 5-5 to 5-8
C_DIR environment variable
  for DOS 1-5
  for HP workstations 4-5
  for OS/2 2-5
  for SPARCstations 3-6
  for Windows 3.x 1-5
C_MODE environment variable 1-10, 2-7
C_OPTION environment variable
  for DOS 1-6
  for HP workstations 4-6 to 4-7
  for OS/2 2-6
  for SPARCstations 3-7 to 3-8
  for Windows 3.x 1-6
cl500
  defined B-1
  invoking 5-5
COFF 5-6
common object file format, defined B-1

D

documentation vii
DOS
  DOS4GVM environment variable 1-7 to 1-9
  installing the tools 1-3
  memory requirements 1-2
  PMINFO A-3 to A-10
  setting up the environment 1-4 to 1-10
  system requirements 1-2
  virtual memory, defined B-4
  virtual-memory management (VMM) 1-7 to 1-9, 1-11
DOS/16M, defined B-2
DOS/4G
  defined B-1
  error messages A-9 to A-10
DOS/4GW
  defined A-1, B-2
  error messages A-1 to A-10
DOS4GVM environment variable 1-7 to 1-9

E

enhancements
  assembler 6-3
  compiler 6-5
  hex conversion utility 6-4
  linker 6-3

environment variables
defined B-2
for DOS 1-4 to 1-10
for HP workstations 4-4 to 4-7
for OS/2 2-4 to 2-7
for SPARCstations 3-5 to 3-8
for Windows 3.x 1-4 to 1-10
error messages
DOS/4G A-9 to A-10
kernel A-5 to A-8
example
assembler 5-2 to 5-3
compiler 5-5 to 5-7
linker 5-4 to 5-6
PMINFO A-3

G

global symbols, defined B-2

H

HP workstations
installing the tools 4-3
setting up the environment 4-4 to 4-7
system requirements 4-2

I

initialized sections, defined B-2
installation
for DOS 1-3
for HP workstations 4-3
for OS/2 2-3
for SPARCstations 3-3 to 3-4
for Windows 3.x 1-3
interlist utility 5-7
defined B-2
invoking
assembler 5-2 to 5-4
compiler 5-5 to 5-8
linker 5-2

K

K&R, related document vii
kernel error messages A-5 to A-8

L

linker, defined B-2
linker walkthrough 5-2 to 5-4
listing file, defined B-2
lnk500
defined B-3
invoking 5-3

M

map file, defined B-3
memory requirements
for DOS 1-2
for OS/2 2-2
for Windows 3.x 1-2
mounting the CD-ROM
for HP workstations 4-3
for SPARCstations 3-3

O

optimizer, defined B-3
OS/2
installing the tools 2-3
memory requirements 2-2
setting up the environment 2-4 to 2-7
system requirements 2-2

P

PATH statement
for DOS 1-4
for OS/2 2-4
for Windows 3.x 1-4
path statement
for HP workstations 4-4
for SPARCstations 3-5
performance considerations 1-11
PMINFO A-3 to A-10
pragma, defined B-3
protected mode
environment, troubleshooting A-2
programs, defined B-3

R

real mode, defined B-3
related documentation vii
release notes 6-1 to 6-10
resetting environment variables
  for DOS 1-10
  for HP workstations 4-7
  for OS/2 2-7
  for SPARCstations 3-8
  for Windows 3.x 1-10

S

section, defined B-3
SPARCstations
  installing the tools 3-3 to 3-4
  setting up the environment 3-5 to 3-8
  system requirements 3-2
static variable, defined B-3
string table, defined B-4
subsection, defined B-4
swap file, defined B-4
symbolic debugging, defined B-4
system requirements
  for DOS 1-2
  for HP workstations 4-2
  for OS/2 2-2
  for SPARCstations 3-2
  for Windows 3.x 1-2

T

.text section, defined B-4
TMP environment variable
  for DOS 1-9
  for HP workstations 4-7
  for OS/2 2-7
  for SPARCstations 3-8
  for Windows 3.x 1-9

U

uninitialized sections, defined B-4
unmounting the CD-ROM
  for HP workstations 4-3
  for SPARCstations 3-4

V

virtual memory, defined B-4
virtual-memory management (VMM)
  performance considerations 1-11
  using the DOS4GVM environment variable 1-7 to 1-9

W

walkthrough
  assembler 5-2 to 5-4
  C compiler 5-5 to 5-8
  linker 5-2 to 5-4
Windows 3.x
  installing the tools 1-3
  memory requirements 1-2
  setting up the environment 1-4 to 1-10
  system requirements 1-2