TAP5000
Translation Assistant Program
User's Guide

Literature Number: SPRU297
October 1998

IMPORTANT NOTICE

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

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

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

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

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

Preface

Read This First

About This Manual

This book explains how to use the TAP5000 translation assistant program to translate code you already have for TMS320C5x devices into code that will run on TMS320C54x devices.

How to Use This Manual

This book is divided into four main chapters:

- ❏ Chapter 1, Introduction, gives you an overview of the translation assistant and associated assembly language development tools.
- ❏ Chapter 2, Using TAP5000, contains detailed information about TAP5000 usage. This chapter explains how to invoke the translation assistant program and use the graphical user interface to aid in the translation process.
- ❏ Chapter 3, General Operation of TAP5000, describes the concepts and assumptions used by the translation assistant program.
- ❏ Chapter 4, Translation Issues, provides detailed information on the issues that affect the translation of ‘C5x code to ’C54x code.

Notational Conventions

This document uses the following conventions:

- The TMS320C54x core is also referred to as ’C54x. The TMS320C5x is also referred to as ‘C5x.
- Program listings, program examples, and interactive displays are shown in a `special typeface`.
- Square brackets ([and]) identify an optional parameter. If you use an optional parameter, you specify the information within the brackets. Unless the square brackets are in a **bold** typeface, do not enter the brackets themselves.
- In assembler syntax statements, column 1 is reserved for the first character of a label or symbol. If the label or symbol is optional, it is usually not shown. If it is a required parameter, it is shown starting against the left margin of the shaded box, as in the example below. No instruction, command, directive, or parameter, other than a symbol or label, can begin in column 1.

symbol **.usect** “*section name*”, *size in words* [, *alignment*]

The *symbol* is required for the .usect directive and must begin in column 1. The *section name* must be enclosed in quotes and the parameter *size in words* must be separated from the *section name* by a comma. The *alignment* is optional and, if used, must be separated from the size by a comma.

Related Documentation From Texas Instruments

The following books describe the ‘C5x and ’C54x 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.

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

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

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, 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.

Trademarks

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

To Help Us Improve Our Documentation . . .

If you would like to make suggestions or report errors in documentation, please send us mail or email. Be sure to include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

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

Email: dsph@ti.com

If You Need Assistance . . .

❑ World-Wide Web Sites

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

❑ North America, South America, Central America

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

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

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:

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

❑ Asia-Pacific

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

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

❑ Japan

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

❑ Documentation

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

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

Email: dsph@ti.com

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

Contents

Chapter 1

Introduction

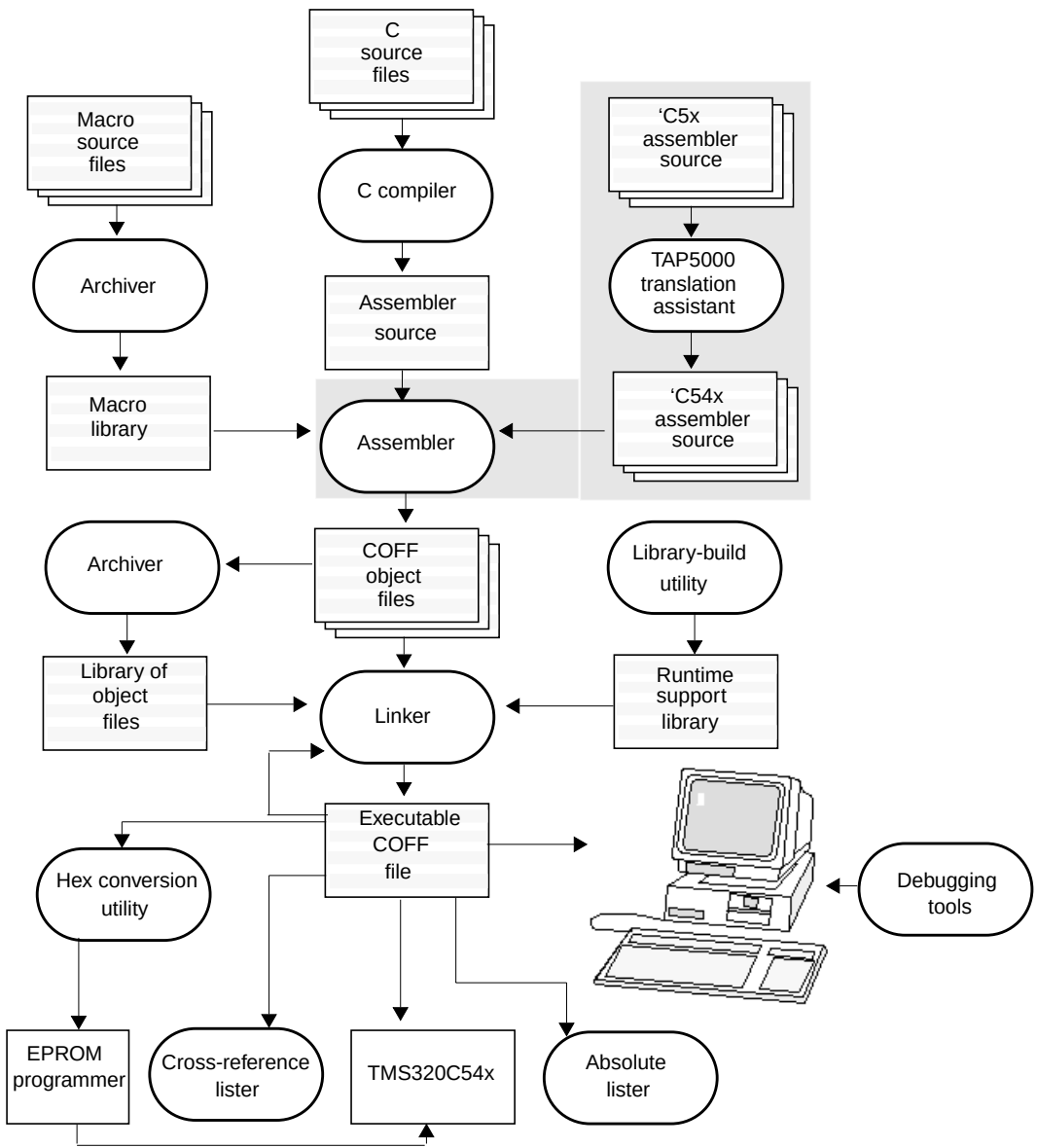
The TAP5000 translation assistant program simplifies the process of converting existing ‘C5x assembly code to ’C54x assembly code. This chapter shows how TAP5000 fits into the general software tools development flow.

Topic **Page**

1.1 Software Development Tools Overview

Figure 1–1 shows the software development flow. The shaded portion highlights the translation assistant program usage.

Figure 1–1 TMS320C54x Software Development Flow



1.2 Tools Descriptions

The following list describes the tools that are shown in Figure 1–1:

- The **C compiler** accepts C source code and produces TMS320C54x assembly language source code. A **shell program**, an **optimizer**, and an **interlist utility** are included in the compiler package:
  - The shell program enables you to compile, assemble, and link source modules in one step.
  - The optimizer modifies code to improve the efficiency of C programs.
  - The interlist utility interlists C source statements with assembly language output to correlate code produced by the compiler with your source code.

  See the *TMS320C54x Optimizing C Compiler User's Guide* for more information.

- The **assembler** translates assembly language source files into machine language COFF object files. Source files can contain instructions, assembler directives, and macro directives. You can use assembler directives to control various aspects of the assembly process, such as the source listing format, data alignment, and section content. See the *TMS320C54x Assembly Language Tools User's Guide* for more information about using the assembler. See the *TMS320C54x DSP Reference Set* for detailed information on the assembly language instruction set.

- The **linker** combines object files into a single executable COFF object module. As it creates the executable module, it performs relocation and resolves external references. The linker accepts relocatable COFF object files (created by the assembler) as input. It also accepts archiver library members and output modules created by a previous linker run. Linker directives allow you to combine object file sections, bind sections or symbols to addresses or within memory ranges, and define or redefine global symbols. See the *TMS320C54x Assembly Language Tools User's Guide* for more information.

- The **archiver** allows you to collect a group of files into a single archive file, called a library. For example, you can collect several macros into a macro library. The assembler searches the library and uses the members that are called as macros by the source file. You can also use the archiver to collect a group of object files into an object library. The linker includes in the library the members that resolve external references during the link. The archiver allows you to modify a library by deleting, replacing, extracting, or adding members. See the *TMS320C54x Assembly Language Tools User's Guide* for more information.

- You can use the **library-build utility** to build your own customized runtime-support library. See the *TMS320C54x Optimizing C Compiler User's Guide* for more information.
- The **hex conversion utility** converts a COFF object file into TI-Tagged, ASCII-hex, Intel, Motorola-SE, or Tektronix object format. The converted file can be downloaded to an EPROM programmer. See the *TMS320C54x Assembly Language Tools User's Guide* for more information.
- The **absolute lister** accepts linked object files as input and creates .abs files as output. You assemble the .abs files to produce a listing that contains absolute addresses rather than relative addresses. Without the absolute lister, producing such a listing would be tedious and would require many manual operations. See the *TMS320C54x Assembly Language Tools User's Guide* for more information.
- The **TAP5000 translation assistant program** accepts 'C5x assembly source input and produces a 'C54x assembly source file. Instructions that are not directly translatable are flagged with error and warning messages to help you complete the translation.
- The **cross-reference lister** uses object files to produce a cross-reference listing showing symbols, their definition, and their references in the linked source files. See the *TMS320C54x Assembly Language Tools User's Guide* for more information.
- The main product of this development process is a module that can be executed in a **TMS320C54x** device. You can use one of several debugging tools to refine and correct your code. Available products include:
  - An instruction-accurate and clock-accurate software simulator
  - An XDS emulator
  - An evaluation module (EVM)

  For information about these debugging tools, see the *TMS320C54x C Source Debugger User's Guide.*

Chapter 2

Using TAP5000

The TAP5000 translation assistant program simplifies the process of converting ’C5x assembly code to ’C54x assembly code. This chapter explains how to use the translation assistant effectively.

Topic **Page**

2.1 TAP5000 Overview

The TAP5000 translation assistant program automates and simplifies the task of converting a working 'C5x application to a ’C54x application. TAP5000 is an interactive tool that creates as complete a translation as possible via automatic analysis/transformation and user intervention. The goal of TAP5000 is to preserve the source code investment by finding sensible translations when possible, and seeking help from the user when not possible.

When moving an existing application to a different platform, many issues arise. System-specific issues include memory organization, external interfacing, and O/S interfacing. Device-specific issues include peripherals, I/O, timing, and emulation. CPU-specific issues include the instruction set, register file, and pipeline behavior. TAP5000 addresses a subset of the CPU-specific issues.

The Texas Instruments applications document, “’5x to ’54x Code Translation”, provides background information on ’C5x to ’C54x translation issues. This user’s guide builds on the issues discussed in the applications document. The applications document is provided at the following web site:

http://www.ti.com/sc/docs/dsps/tools/ftools/index.htm

Note: Effective Use of the Translation Assistant

Effective use of this translation assistant requires an understanding of the ’C5x and ’C54x architectures and instruction sets and the ’C5x assembly application being translated.

2.1.1 TAP5000 Features

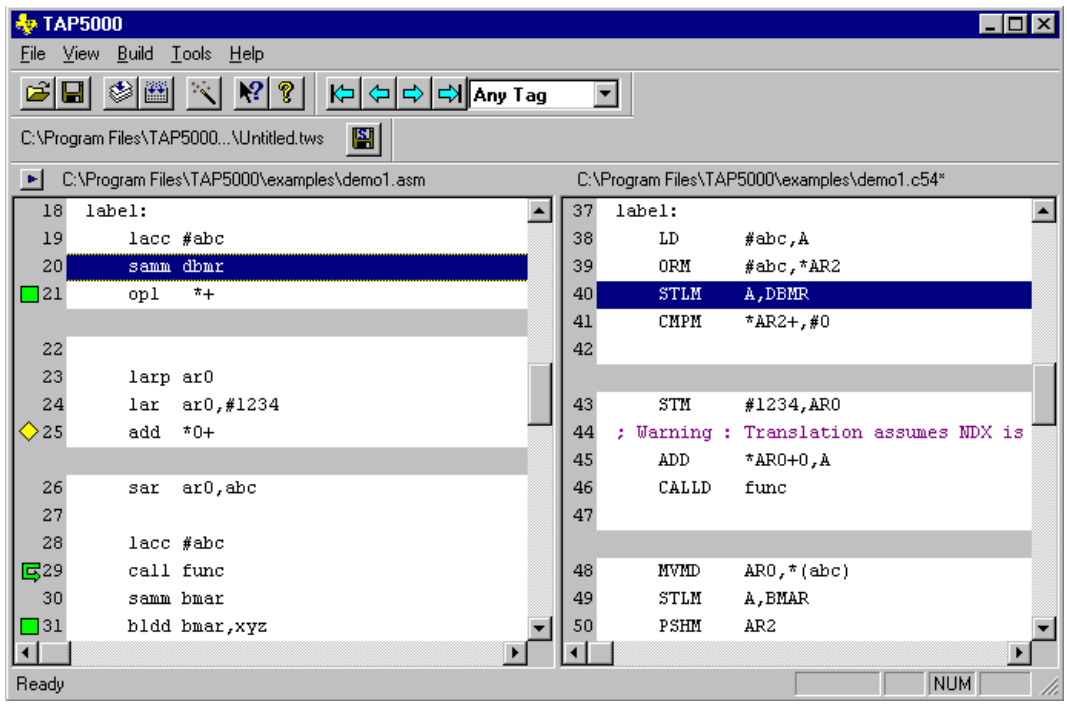
TAP5000 includes the following features:

- ❏ a side-by-side display of source code and translated output
- ❏ context-sensitive analysis that allows more complete and efficient translations than line-by-line translators
- ❏ icons to visually flag translation issues/problems in the display
- ❏ interactive controls that allow many translation problems to be resolved on the spot, with immediate feedback
- ❏ the preservation (as much as possible) of comments, symbols, and source code formatting
- ❏ the scheduling of translated code to avoid unprotected pipeline latencies, so that the code can operate correctly in the ’C54x pipeline.

2.1.2 TAP5000 Interface and Files

Figure 2–1 displays the TAP5000 interface.

Figure 2–1 Overview of the TAP5000 Interface



The Source window (the left window) displays the ’C5x input file. ’C5x input files should assemble correctly with the 'C1x/'C2x/'C5x assembler run with the -v50 switch. ’C2x-style mnemonics are also accepted.

The Source window also includes icons that mark translation issues. To respond to a translation issue, simply double-click on the icon. For more information, see Section 2.5.1, *Responding to a Translation Issue Via the Translation Wizard*, page 2-9.

The Translation window (the right window) displays the translated output file. TAP5000 produces an output file containing TMS320C54x mnemonic assembly source code. The file has a .c54 extension. TAP5000 will not produce algebraic assembly code.

The translated output file may contain (as comments) errors or warnings related to potential translation problems.

The Source and Translation windows are aligned, and scrolling is synchronized. Clicking on any source code line will highlight the corresponding line(s) in the Translation window.

TAP5000 uses the same assembly source parser as the ’C5x assembler. If the source file requires ’C5x assembler options to assemble, you can specify these options before translating the file. See Section 2.4.1, *Setting ’C5x Assembler Options*, page 2-7 for more information.

TAP5000 will not translate an input file that will not assemble with the ’C5x assembler. If TAP5000 encounters any errors or warnings while parsing the ’C5x source file, it displays a dialog box to inform you that the translation is being aborted. The errors are reported in the Output window. This window appears when an error or warning is detected in the ’C5x assembler source, if it isn’t already open.

2.1.3 TAP5000 Limitations

TAP5000 has the following limitations:

- ❏ Some ’C5x instructions have no practical translation. These instructions are flagged as errors and translation is not attempted.
- ❏ TAP5000 translates the instruction stream to a nearly equivalent set of instructions. It cannot restructure the application to take advantage of ’C54x features.

  For example, you must edit the translated code accordingly to account for (and take advantage of) the differences in the ’C5x and ’C54x memory architectures. Likewise, if you wish to take advantage of the ’C54x calling conventions, you must modify the translated code accordingly.
- ❏ TAP5000 does not address issues relating to memory maps, peripherals, hardware configuration, or other system-level characteristics.
- ❏ TAP5000 has no editing capabilities.
- ❏ TAP5000 cannot translate macro definitions or conditionally assembled code.
- ❏ TAP5000 makes certain assumptions about the control and data flow of the program. If these assumptions are not true in the source code, the translation may be incorrect.

2.2 Translation Flow

’C5x to ’C54x translation requires the following steps:

1) Open each source file within the TAP5000 interface to produce an initial translation. For more information, see Section 2.4, *Translating Source Files*, page 2-6.

2) Provide information to TAP5000 when needed. TAP5000 includes various methods for you to provide information and to help in the translation process. For more information, see Section 2.5, *Handling Translation Issues*, page 2-8.

3) Save the translation output file. For more information, see Section 2.9, *Saving the Translated File*, page 2-15.

4) Handle any remaining translation issues, including those highlighted by TAP5000, by editing the output file as necessary. For more information, see Section 2.11, *Using Translated Output Files in a ’C54x Application*, page 2-16.

5) Debug the application as necessary to handle any final issues.

While the resulting translation from TAP5000 may not be complete, it is an excellent first pass and saves considerable time.

2.3 Invoking TAP5000

TAP5000 is a PC application. To invoke TAP5000, use one of the following methods:

- ❏ Double-click its shortcut icon in the desktop, or
- ❏ Click the Start button, and then point to Programs. In the Programs listing, point to TAP5000 v1.00, and then click TAP5000.

2.4 Translating Source Files

To translate a source file within the TAP5000 interface:

1) From the File menu, select Open.
2) Locate and select the desired source file to be translated.
3) Click OK.

Translation issues are flagged with icons in the Source window. You should respond to every translation issue. For more information on responding to translation issues, see Section 2.5.1, *Responding to a Translation Issue Via the Translation Wizard*, page 2-9. Any changes made in response to translation issues will result in an automatic re-translation of the source file.

You can translate at any time by selecting the Translate option from the Build menu. To re-translate the initial source code of a file, ignoring any changes that have been made to it in TAP5000, select the ReTranslate option from the Build menu.

2.4.1 Setting ’C5x Assembler Options

TAP5000 uses the same assembly source parser as the ’C5x assembler. If your source file requires ’C5x assembler options to assemble without errors, you can specify the options as follows:

1) From the Tools menu, select Input File Settings.

2) In the Input File dialog, fill in the option fields as needed. TAP5000 allows you to specify the assembler’s -c, -d, -i, and -u options. For more information on these options, see the *TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User's Guide.*

2.4.2 Translating Include Files

In general, an include file that is independent of context (i.e., can be assembled on its own without errors) can be translated individually like any other file. However, if an include file is dependent upon the context of the file in which it is included (the parent file), TAP5000 allows you to translate the file within that context.

For example, if a file named parent.asm defines a symbol that is used by an include file named include.asm:

1) Translate the parent file (parent.asm) as you would any other file.

2) In the source code window, right-click on the line that includes the include file (include.asm) to display the context menu.

3) From the context menu, select the Open Include File option.

The include file is then translated within the context of the parent file.

To close an include file, select Close Include File from the context menu. You will be prompted to save the translation of the include file. The name that you give the translated include file will be incorporated into the translated parent file. For example, if the translated include file is saved as "include.c54", the translated parent file will contain a line that says ".include include.c54".

If an include file is referenced in more than one file, its initial translation is most likely correct for use within the additional parent files. However, you may need to edit the other translated parent files to ensure that the name of the include file is accurate.

2.5 Handling Translation Issues

TAP5000 informs you of possible translation issues by marking problematic lines with one of the icons described below.

Red octagon	(Error) Indicates a situation where the translation is known to be incorrect, impossible, or beyond the limits of the translation assistant program. In some cases, you may be able to supply information that allows a correct translation to be produced.
Yellow diamond	(Warning) Indicates a situation where the translation is possibly correct, but depends on a particular assumption. You have the opportunity to confirm or deny the assumption.
Green square	(Efficiency hint) Indicates a situation where the translation is correct, but could be improved with additional information. You have the opportunity to provide that information.
Green arrow	(Linkage point) Indicates a situation (such as a call, branch, or label) where program control may transfer to or from non-local parts of the program. Correct translation of the code near a linkage point depends on knowing whether one or more registers is “live” at that point in the program. A register is live if it currently contains a value that will subsequently be used in another instruction. For more information, see Section 3.5.1, *Liveness Settings at Linkage Points*, page 3-9.
Checkmark	(Assertion) Indicates that you’ve responded to a translation issue for this line of code. To view or change your response, simply double-click on the checkmark.

Double-clicking on a source line marked with an icon invokes the Translation Wizard. This dialog explains the translation issue(s) with the selected line and allows you to respond.

Sometimes a given source line may have multiple issues or comments associated with it. In these cases, the Translation Wizard displays a tabbed page for each issue or comment.

2.5.1 Responding to a Translation Issue Via the Translation Wizard

Within the Translation Wizard, you may be asked to:

- ❏ provide information that cannot be determined automatically by analyzing the program, or
- ❏ confirm or deny the validity of assumptions made by TAP5000 when potential translation problems occur.

After you respond, the Source and Translation windows are automatically updated to reflect changes to the translation.

A line that has been affected by your response is marked with a checkmark.

A Translation Wizard may include an “Apply to Default Settings” option. When you use this option, your response to the translation issue becomes the default behavior for all lines affected by this issue:

- ❏ The global translation settings or calling convention settings are updated according to the new setting. For more information, see Section 2.6, *Setting Global Translation Defaults*, page 2-12 or Section 2.7, *Setting Global Calling Convention Defaults*, page 2-13.
- ❏ Any line affected by this issue is translated according to the new setting, unless you’ve already responded to this issue for a particular line. A new global setting will not override any previously-set local response.

To revert to the global behavior for a particular translation issue, use the Reset button in the Translation Wizard.

To change or undo your response to a particular issue, open the Translation Wizard (by double-clicking on the appropriate checkmarked line) and change your response accordingly.

2.5.2 Setting Register Usage Information at Linkage Points

The analysis of register usage, known as register liveness, is critical for correct and efficient translation. A register is considered live if it currently contains a value that will be used subsequently by other instructions. It is very important for you to specify register liveness at linkage points. Linkage points are marked with green arrow icons in the Source window.

To specify the registers that are in use at a linkage point:

1) Double-click on a source line with a linkage point to open the Translation Wizard dialog box.
2) In the Translation Wizard dialog, select registers on each tabbed page, as necessary. For specific information about the tabbed page, click the Help button. For information on any register or button within the tabbed page, click the ? icon (in the top right corner of the dialog), then click the item.
3) Click OK.

For calls, you must specify the registers that are live at the entry and exit points of the call, and you must also specify the registers that are preserved across the call.

When a register is live across a linkage point, it requires two settings: one at the point it is written and another at the point it is read. These settings should be mutually consistent, but TAP5000 cannot and does not check for this.

For more information on linkage points, see Section 3.3, *Linkage Points*, page 3-5. For more information on register liveness, see Section 3.5, *Analysis of Register Usage*, page 3-8.

2.5.3 Stepping Through a File

You can move quickly from issue to issue within a file by clicking one of the arrow icons on the browse toolbar. Use the drop-list on the browse toolbar to select the type of tag to search for: errors, warnings, assertions, hints, or linkage points.

To move to the first translation tag of the specified type, click:

To move to the previous translation tag of the specified type, click:

To move to the next translation tag of the specified type, click:

To move to the last translation tag of the specified type, click:

2.6 Setting Global Translation Defaults

TAP5000 allows you to specify certain global information that can be used by the translation assistant. For example, you can specify the use of compatibility mode, assumptions about status bits, or the instruction scheduling method. This information is helpful because it can be used each time a particular translation issue is encountered.

You can set global defaults in the Translation dialog box:

1) From the Tools menu, select Translation Settings.
2) In the Translation dialog box, set options on each tabbed page, as necessary. For specific information about the tabbed page, click the Help button. For information on any item within the tabbed page, click the ? icon (in the top right corner of the dialog), then click the item.
3) Click OK.

To reset the default settings on a page, click Reset. To close the dialog box without saving any changes, click Cancel.

If you load a global settings file, it will overwrite the current global settings. For more information on saving and loading a global settings file, see Section 2.8, *Saving Global Default Settings*, page 2-14.

Global settings can also be specified within the Translation Wizard. If you provide information in the Translation Wizard that can be applied to other instances of the translation issue, click on Apply to Default Settings.

Note: Global settings do not override local settings.

Global settings will not override any setting that was previously applied to a particular source line. To change such a source line to use the global settings, double-click on the line to open the Translation Wizard. Then click the Reset button.

You can also set global calling convention defaults. For more information, see Section 2.7, *Setting Global Calling Convention Defaults*, page 2-13.

2.7 Setting Global Calling Convention Defaults

If your program follows certain conventions for register use at function calls and returns, you can set this information globally. This information is useful in the translation process. For more information on how TAP5000 uses register analysis, see Section 3.5, *Analysis of Register Usage*, page 3-8 and Section 3.5.1, *Liveness Settings at Linkage Points*, page 3-9.

You can set these global defaults in the Calling Conventions dialog box:

1) From the Tools menu, select Calling Conventions.
2) In the Calling Conventions dialog box, set options on each tabbed page, as necessary. For specific information about the tabbed page, click the Help button. For information on any item within the tabbed page, click the ? icon (in the top right corner of the dialog), then click the item.
3) Click OK.

To reset the default settings on a page, click Reset. To close the dialog box without saving any changes, click Cancel.

Global settings can also be specified within the Translation Wizard. If you provide information in the Translation Wizard that can be applied to other instances of the translation issue, click on Apply to Default Settings.

Note: Global settings do not override local settings.

Global settings will not override any setting that was previously applied to a particular source line. To change such a source line to use the global settings, double-click on the line to open the Translation Wizard. Then click the Reset button.

2.8 Saving Global Default Settings

To save all of the global default settings (input file, translation, and calling convention settings) for use in other translation sessions, select the Save Settings option from the File menu. The information is saved in a file with a .tws extension. Saving the default settings is useful when your application has several files, and you want to use the same global settings for each file.

To load a global settings file previously created:

1) From the File menu, select Open Settings.
2) Select the appropriate .tws file.
3) Click OK.

If you load a global settings file, it will overwrite the current global settings.

You also can automatically load a global settings file at startup:

1) From the Tools menu, select Display Options.
2) Click the Workspace tab.
3) Select Reload Last Settings File at Startup.

When you next invoke TAP5000, the last settings file used will be reloaded.

2.9 Saving the Translated File

To save the ’C54x source file generated by TAP5000, from the File menu, select Save—Save Translated Output.

By default, the translated file has a .c54 extension.

2.10 Saving the Translation Session

To save the work you’ve done in a translation session, from the File menu, select the Save—Save Translation State option. This command saves the “state” of the translation (settings, changes to the translated output file, etc.) in a file with a .ir extension. This functionality is useful if you need to exit TAP5000, but you want to pick up later where you left off.

To open the translation state file at a later time:

1) From the File menu, select Open.
2) From the Open dialog box, select the appropriate .ir file. You may have to change the Files of Type field in order to view the .ir file(s).
3) Click OK.

2.11 Using Translated Output Files in a ’C54x Application

In order to use the ’C54x output files generated by TAP5000, you must:

1) Correct all translation errors. You should address all lines that are flagged with errors and warnings by TAP5000. In addition, it may be necessary for you to further modify the translated code to address issues that are beyond the scope of TAP5000. For more information on these issues, see

2) Assemble the translated code with the ’C54x assembler (asm500).

3) Assemble c5xregs.asm (provided in the TAP5000 installation directory) with asm500 and link it with the rest of your application. TAP5000 uses dedicated memory locations to model some of the ’C5x registers not present on the ’C54x. The file c5xregs.asm defines and allocates these registers. For more information on the memory-modeled registers, see Section 3.7.2, *Registers Mapped to Memory Locations*, page 3-14.

4) Modify your system initialization code by adding the following instructions:

```
RSBX  CMPT      ; clear ARP compatibility mode
RSBX  CPL       ; clear compiler mode
STM   #0,ZERO    ; initialize translator’s ZERO location
```

2.12 Accessing Online Help

Online help is available to provide information about menu options, toolbar icons, dialog boxes, and windows.

To display a list of help topics, from the Help menu, select Help Topics. Double-click the topic that you want to view.

To find out about an item in the TAP5000 display:

1) Click the Help icon on the toolbar:

This changes the pointer to a question mark.

2) Select the menu option or click the item that you want more information about.

To find out about the active dialog box, click the Help button in that dialog box.

For help on an item within a dialog box:

1) Click the question mark icon at the top right corner of the dialog box:

2) Then click on the item.

Chapter 3

General Operation of TAP5000

The TAP5000 translation assistant program relies on a combination of analysis, interaction, and transformations to overcome various translation issues. This chapter provides information on resource assignments, global assumptions, and analysis concepts behind the operation of TAP5000.

Topic **Page**

3.1 Assumptions About the Source Program

TAP5000 performs various analysis and transformation steps to produce a complete and efficient translation. Many of these steps are similar to the steps performed by a compiler in translating a high-level language to assembly object code.

The correctness of these steps depends in part on certain assumptions made by TAP5000 about the source program. Code that violates these assumptions may not translate correctly. TAP5000 makes the following assumptions:

- **No branches to unlabeled instructions.** This assumption applies especially to indirect branches (BACC). TAP5000 assumes that any possible destination of such a branch is labeled. If TAP5000 encounters a direct branch with an expression operand (for example, "B label+3"), it evaluates the expression and inserts a translator-generated label at the appropriate location.
- **No indirect writes to MMRs.** The ’C5x register set is accessible via memory locations on page 0. TAP5000 relies on being able to detect all instances in which these registers are written. It can detect direct writes by instructions, writes using the memory-mapped mode instructions (SAMM), or writes using absolute addresses on page 0. However, TAP5000 cannot detect writes in any of the following ways:
  - Via an indirect operand, such as `SACL *`
  - Via a direct symbolic operand, such as `SACL abc`
  - Via a direct absolute operand, unless TAP5000 can detect page 0 by the presence of a nearby `LDP #0`
- **No out-of-order execution.** The ’C5x has a very small number of cases of unprotected pipeline latencies. In other words, the hardware will not stall the pipeline to insure that a value has been completely written before a subsequent instruction tries to read it. In such a case, it is possible to "trick" the pipeline by placing an instruction after the write that relies on the fact that the new value has not been written yet. Code written in this way is not interruptible (even on the ’C5x) and will not be correctly translated. For example:

  ```
  SAR AR3, INDX  ; write AR3 to INDX
  MAR *0+     ; expects INDX to still contain old value
  MAR *0+    ; INDX now contains AR3
  ```

  This code will not translate correctly.
- **No handling code as data, or data as code.** TAP5000 does not support practices such as self-modifying code, or .word directives that define executable instructions and alter the program’s state.

In addition, TAP5000 depends on accurate register liveness information at linkage points in order to correctly analyze the program. It is your responsibility to provide this information. See Section 3.3, *Linkage Points*, page 3-5 and Section 3.5, *Analysis of Register Usage*, page 3-8.

3.2 Control Flow Analysis of Functions

To begin the translation process, TAP5000 separates the code in a source file into disjoint, logical units that connect at interface points called linkage points. These units are referred to as functions.

In general, the code is divided along COFF section boundaries. In most cases, there is one function per COFF section. Each COFF section may be further subdivided if TAP5000 detects disjoint functions within the section. Disjoint functions are regions that are not connected by branches (although they can call each other).

After the code has been divided into functions, each function is analyzed separately to determine its control flow, and translated independently of the other functions. TAP5000 assumes that control can transfer from one function to another only at linkage points. For more information on linkage points, see Section 3.3, *Linkage Points*, page 3-5.

The individual functions and their control flow analysis provide the basis for the TAP5000 translation process.

In spite of the fact that TAP5000 internally divides the code in this way, both the Source window and Translation window retain the original source order. For example, if two COFF sections are intermixed in the source file, they will be displayed as intermixed, even though TAP5000 analyzes them separately.

3.3 Linkage Points

Control flow analysis is limited to individual functions. However, at certain points, control may transfer from one function to another. These points, called linkage points, affect the translation process. A linkage point is an entry point, an exit point, or a call site.

An **entry point** is any point at which control may enter a given function. For example:

- ❏ the first instruction in a function
- ❏ any globally visible (via .global or .def) labels defined in the function
- ❏ labelled instructions that can be reached via indirect branches or calls
- ❏ regions of code that appear to be otherwise unreachable

An entry point is marked in the Source window with the following green arrow:

An **exit point** is any point at which control may exit a given function. For example:

- ❏ the last instruction in a function
- ❏ branches to other functions or externally visible (via a .global or .ref) labels
- ❏ indirect branches
- ❏ return instructions

An exit point is marked in the Source window with the following green arrow:

A **call site** is a point at which control exits the function and returns at the same point. For example:

- ❏ a call
- ❏ a trap
- ❏ a software interrupt

A call site is marked in the Source window with the following green arrow:

TAP5000 assumes that control can transfer from one function to another only at linkage points. Furthermore, TAP5000 assumes that control transfer at a linkage point could occur to or from any other function in the program, including functions in other source modules. This assumption is conservative,

since some functions may not be reachable from outside of the module in which they're defined.

TAP5000 cannot analyze register usage across linkage points. The analysis of registers and their usage, or liveness, is critical for correct and efficient translation. Consequently, it is very important for you to provide register liveness information at each linkage point. Register liveness and how it affects the translation of your program is discussed in detail in Section 3.5, *Analysis of Register Usage*, page 3-8.

TAP5000 also allows you to specify the ARP value at entry points. If your program uses a convention for the ARP value at entry points, you can set this value globally in the Calling Conventions dialog. For more information on setting the value of ARP, see Section 4.10, *Supplying the Value of ARP*, page 4-12.

3.4 Tracking Register Values

Within each individual function, TAP5000 tracks the values contained in registers at various points. Value tracking is helpful in cases where a ’C5x register has no analogous ’C54x register. If TAP5000 can determine the value in the register, that value can be substituted directly, rather than having to model the missing register in some way.

For example, knowing the value of the ’C5x ARP register allows TAP5000 to convert ’C5x ARP-based indirect addressing to ’C54x explicit AR addressing. For more information on ARP handling, see Section 4.10, *Supplying the Value of ARP*, page 4-12.

TAP5000 attempts to keep track of the contents of the following ’C5x registers: ARP, BMAR, DBMR, DP, and various bits in status registers ST0 and ST1.

TAP5000 tracks DP for the sake of detecting writes to memory-mapped registers. On the ’C5x, registers can be written to by setting DP to 0. For example, the instruction:

```
SACL DBMR
```

can write to DBMR if DP has been set to 0 previously. If TAP5000 couldn’t track the DP in this case, the translation of DBMR might be incorrect.

TAP5000 can track registers, including the ARP, through both straight-line code and PC discontinuities. TAP5000 cannot track registers across linkage points, as discussed in Section 2.5.2, *Setting Register Usage Information at Linkage Points*, page 2-10.

At a call site, you can assert that a register is preserved across the call, via the Preserved Registers tab of the Translation Wizard. In this case, it can be tracked. For example, by default, TAP5000 will issue an error for the following code:

```
MAR *, AR1  ; sets ARP to AR1
CALL f      ; presumed to "kill" ARP
LACC *       ; error: ARP unknown
```

To eliminate this error and help the translation process:

1) Double-click the source line that contains the call site to open the Translation Wizard dialog.
2) Click the Preserved Registers tab.
3) Select ARP from the list of registers. (When the register is selected, a checkmark appears next to it.)

3.5 Analysis of Register Usage

The analysis of registers and their usage, or their liveness, is critical for correct and efficient translation. A register is considered live if it currently contains a value that will be used subsequently by other instructions. If it does not, the register is considered to be dead. Detection of dead registers is useful for two reasons:

- This knowledge determines whether certain side effects of particular instructions are important. It is unnecessary to write to certain registers (including status bits) in the translation if they are known to be dead.

  For example, in certain instances, TC does not have to be set in the translation if the register is known to be dead.

- Dead registers can be used as temporary registers to help translate other instructions. See Section 3.6, *Use of Temporary Registers*, page 3-13.

Note: If an instruction appears to have no effects, it does not get translated.

If TAP5000's register liveness analysis shows that an instruction has no effects, the instruction will not be translated. If instructions seem to be missing from your translated output, you may need to supply liveness information (at linkage points or globally) to inform TAP5000 that the results of the instructions will be used.

A register is considered live between the point at which it is written and the point at which it is used for the last time (before it is written again). To determine the liveness of a register, TAP5000 starts at a usage point, then scans backwards to find its corresponding write. In the most basic situation, a register is written and used only within the same function.

However, if a register is written in one function and used in another, its liveness cannot be tracked. In this situation, the register is live across a linkage point. Because TAP5000 analyzes functions independently, it has no way to accurately determine liveness across linkage points.

Consequently, it is very important to specify register liveness at linkage points.

3.5.1 Liveness Settings at Linkage Points

TAP5000 provides five types of register usage settings:

- **For a call site (a call, trap or software interrupt instruction):**
  - Register Inputs. These registers are written by the caller and passed to the called function as arguments. Your settings characterize the code above the call site.
  - Register Outputs. These registers are written by the called function and returned to the caller as return values. Your settings characterize the code below the call site.
  - Preserved Registers. These registers are not changed by the call. (Either they're not used in the called function, or the called function preserves them by saving and restoring them). Your settings characterize the code above and below the call site.

  Note that call site settings affect only the code at the call site. They do not affect the settings for the called function itself. These settings must be specified at the entry and exit points of the called function.
- **For an entry point (first instruction, or globally visible label):**
  - Register Inputs. These registers have been written elsewhere and will be read by this function. Your settings characterize the code below the entry point.
- **For an exit point (return instruction, or branch to externally visible label):**
  - Register Outputs. These registers are written in this function and are expected to be read at the destination point. Your settings characterize the code above the exit point.

Note that entry and exit point settings affect only the code in their own function. They do not affect the settings for the caller, which must be specified at the corresponding call sites.

3.5.2 Default Liveness Conventions

If a calling convention results in common register usage at most or all linkage points, you can specify these settings in the Calling Conventions dialog box. You can override them at specific linkage points if necessary. For more information on setting global register usage conventions, see Section 2.7, *Setting Global Calling Convention Defaults*, page 2-13.

The initial register usage settings used by TAP5000 assume that only the accumulator (ACC) is live across linkage points, and that no registers are preserved by calls.

For more information on linkage points, see Section 3.3, *Linkage Points*, page 3-5.

3.5.3 Dedicated Registers

In some situations, certain registers need to be considered as live at all times. For example:

- ❏ control registers that affect the execution state of the machine
- ❏ registers used in interrupt service routines without being saved and restored

These registers are considered to be "dedicated", and you can specify them on the Dedicated Registers page of the Calling Conventions dialog.

A register specified as dedicated will always appear live to TAP5000. TAP5000 will always model its effects, and will never allocate the register as a temporary register. Note, however, that specifying more dedicated registers than necessary may result in translation errors, if TAP5000 cannot find a dead register to use as a temporary register.

3.5.4 Liveness Examples

The examples in this section illustrate the use of register liveness settings at linkage points.

- Argument Passing. A function places a value in a register and calls another function, which uses that value.

```
    ...
    write reg
    call f  ; call site: set reg as register input to f
    ...
    ...
f:  ... ; entry point: set reg as a register input
    read reg
    ...
```

- Return Values. A function places a value in a register and returns to its caller, which uses that value.

```
   ...
   call f ; call site: set reg as a register output of f
   read  reg
   ...
   ...
f:  ...
   write reg
   ret     ; exit point: set regas a register output
```

- Dedicated registers. Dedicated registers are considered to be live at all times, because they are active at all times.

```
   write reg   ; writes some value
   <interrupt occurs>
ISR:
    use reg    ; specify reg as dedicated
```

  For more information on dedicated registers, see Section 3.5.3, *Dedicated Registers*, page 3-10.

  Specify dedicated registers as live in the Calling Conventions dialog. Setting a register as dedicate prevents TAP5000 from using it as a temporary register. For more information on the Calling Conventions dialog, see Section 2.6, *Setting Global Translation Defaults*, page 2-12.

- "Context Saved" Registers (Preserved). Similar to dedicated registers, preserved registers are expected to remain unchanged. However, they are saved and restored by a function, as in the case of a register used as a stack pointer.

```
   write reg   ; writes some value
   call f      ; call site: set reg as preserved by f
   read reg    ; expects same value
   ...
   ...
f:  ...        ; entry point: set reg as register input
    save reg    ; optional instruction
    ...
    restore reg ; optional instruction
    ret        ; exit point: set reg as a register output
```

  Note that specifying *reg* as an input and output here prevents TAP5000 from removing the save and restore.

- Passing via branch. In this case, a function places a value in a register and branches to another entry point (for example, in a different source module or section).

```
  ...
  write reg
  branch label ; exit point: set reg as a register output
  ...
  ...
f:  ...   ; entry point: set reg as a register input
    read reg
    ...
```

3.6 Use of Temporary Registers

Some translations require a sequence of instructions to correctly model the behavior of a source instruction. In some cases, temporary registers are required as part of these sequences. TAP5000 allocates temporary registers as necessary from the set of ’C54x registers.

TAP5000 tries to use dead registers for temporary registers where possible. (For information on dead registers, see Section 3.5, *Analysis of Register Usage*.) A dead register can be used as a temporary register only if it is the appropriate type of register. For example, a temporary register to be used for an accumulator must be an accumulator register itself.

If TAP5000 is unable to find a dead register to use as a temporary register, it will use a live register of the appropriate type. In this case, its value must be preserved. TAP5000 will save and restore the register’s value via pushes and pops to the stack. Note, however, that saving and restoring registers requires the addition of code to the translated file.

3.7 Register Translations

To translate a program, the source processor's registers must be modeled on the target according to a consistent convention. In general, most of the ’C5x resources map easily to the ’C54x. However, many translation issues result from the ’C5x resources that have no ’C54x equivalent.

3.7.1 ’C5x Registers Mapped to ’C54x Registers

The ’C5x registers mapped to ’C54x registers are described in Table 3–1.

Table 3–1 ’C5x to ’C54x Register Mapping

’C5x register	’C54x register
ACC	A accumulator
ACCB	B accumulator
AR0 - AR7	AR0 - AR7
ARCR	AR0 (if NDX=0)
INDX	AR0 (if NDX=0)
TREG1	T (if NDX=0)
TREG2	T (if NDX=0)
status bits	corresponding bits
PM	FRCT (for PM=0 or PM=1)

3.7.2 Registers Mapped to Memory Locations

Some ’C54x registers do not exist on the ’C54x. These registers are modeled as memory locations. TAP5000 requires these memory locations to be located in the scratch-pad RAM area on page 0 of data memory, where they can be addressed using the memory-mapped register mode of the ’C54x. In addition, TAP5000 reserves one location for specific translations: ZERO, a memory location containing the value 0.

TAP5000 does not always require memory locations to model these registers. It tries to use register tracking, usage analysis, and transformation techniques to avoid using memory to model ’C5x registers.

The list of memory-modeled registers used by TAP5000 is:

ARCR	models ARCR register (if CMPT is 1)
BMAR	models BMAR register
CBER1	models CBER1 register
CBER2	models CBER2 register
CBSR1	models CBSR1 register
CBSR2	models CBSR2 register
DBMR	models DBMR register
INDX	models INDX register (if CMPT is 1)
PM	models the PM (product shift mode) bits
PREG	(2 words) models PREG
TREG1	models TREG1 register (if TRM is 1)
TREG2	models TREG2 register (if TRM is 2)
ZERO	contains the value 0 for translating ADCB, SBBB

Memory-modeled registers are allocated in a separate section called .c50regs, which is defined in the assembly source file c50regs.asm. This file is shipped with TAP5000, and should be assembled and linked into your ’C54x application.

TAP5000 uses .global directives at the top of each translated file to declare the symbols representing memory-modeled registers.

Because the memory locations used to model ’C5x registers are part of the execution state of the program, they must be preserved during context switches (e.g., interrupt routines). TAP5000 does not automatically insert saves and restores of these memory registers.

TAP5000 models the ’C5x memory directly: each ’C5x memory location is modeled by the ’C54x location with the same address. Addresses that refer to ’C5x memory-mapped registers are translated to the corresponding registers on ’C54x. No other address mapping is done; for example, the translation does not detect or account for differences in the ’C5x and ’C54x memory maps.

3.7.3 ’C5x Registers Unmapped in Translation

Several ’C5x registers are not modeled by TAP5000. These registers are primarily tied to specific elements of the ’C5x hardware: processor control registers and peripheral registers. Many of these registers have similar, but not identical, counterparts on the ’C54x. In some cases, the counterparts even have the same names. For example, both the ’C5x and ’C54x have time control registers called TIM. In other cases, the registers don’t have the same names. For example, one of the ’C5x serial ports registers is DRR, while the ’C541 has DRR0 and DRR1.

Because of the device-specific nature of these registers, TAP5000 does not attempt to translate them. Instead, they are left in the translated code under their original names, and a warning is issued. In cases where the ’C54x has no register by that name, the translated code will not assemble until you correct it manually.

3.8 Instruction Scheduling

In some cases, the timing or pipeline behavior of the translated instructions may differ from that of the source. Furthermore, the ’C54x has several cases of unprotected pipeline latencies. In other words, the hardware will not stall the pipeline to insure that a value has been written before a subsequent instruction tries to read it.

TAP5000’s scheduling frequently results in the reordering of translated instructions. It tries to retain the original source order of instructions as much as possible. However, TAP5000 may reschedule the translated code in order to minimize the number of NOPs required. Sometimes, instructions may be reordered for no apparent reason. This reordering is due to the scheduling algorithm used by the instruction scheduler.

The instruction scheduling can be seen by clicking on a source line in the Source window. The corresponding translated instructions are highlighted in the Translation window.

You can change the default method of instruction scheduling by modifying the Scheduling page of the Translation dialog. For information on setting translation defaults, see Section 2.6, *Setting Global Translation Defaults*, page 2-12.

You can force TAP5000 to retain the original order as much as possible with the Try to Preserve Original Order option. In this case, TAP5000 would insert as many NOPs as needed.

You can also choose to have TAP5000 "optimize" the translated code with the Minimize NOPs option. In this case, TAP5000 will ignore the original order and reschedule the translated code as much as possible, even when rescheduling isn’t necessarily needed. It will reschedule to reduce the possibility of NOPs.

TAP5000 tries to schedule all branches and calls as delayed, even if the corresponding source instructions are not. On the ’C54x, delayed branches and calls execute two words (either 2 1-word instructions or 1 2-word instruction) before the control transfer actually occurs. TAP5000 tries to fill these delay slots with useful instructions. If it can’t fill at least one delay slot, it reverts to using the undelayed form of the call or branch. For more information on delay slots, see Section 4.15, *Translation of XC Instructions*, page 4-16.

TAP5000 does not schedule between blocks of code. NOPs may be added to the translated code if TAP5000 can’t determine what is in the next block (e.g., at an exit point) or if the next block hasn’t been scheduled yet (as in a loop).

Chapter 4

Translation Issues

Because of the differences between the ’C5x and ’C54x, there are many issues that affect translation. This chapter describes the specific issues related to ’C5x to ’C54x translation and discusses the strategy used by TAP5000 for dealing with each. This chapter also relates to some of the translation issues that appear in the Texas Instruments applications document, “’5x to ’54x Code Translation”. The applications document is provided at the following web site:

http://www.ti.com/sc/docs/dsps/tools/ftools/index.htm

Topic **Page**

4.1 Uses of PREG

This section corresponds to Section 3.2.2 of the applications document.

TAP5000 can analyze and reorder the instructions in loops. Consequently, a loop can be restructured to better match the processor's instruction set. In particular, this transformation helps TAP5000 to eliminate the use of the 'C5x PREG, which is used to hold products before accumulating them.

'C5x uses PREG for storing the output of the multiplier. 'C54x does not have an analogous feature. The output of the multiplier is transferred to, added to, or subtracted from accumulator A or B in the same cycle in which the output is computed.

In most cases, PREG is a short-lived temporary register that holds a value from the time it is written (i.e, produced by the multiplier) until it is used. The register value is used when it is moved to an accumulator, added to or subtracted from the accumulator, or stored. Certain 'C54x instructions perform the write and use operations in one step, eliminating the need for the temporary register in most cases.

To eliminate the need for PREG, TAP5000 tries to combine the instruction that writes PREG (the multiply) with the instruction that uses it (add/subtract, move, or store) to form the corresponding 'C54x instruction, such as MAC, MAS, or MPY. It relies on being able to pair up the writes and uses. In cases where it cannot do this, TAP5000 models PREG in memory.

Eliminating PREG in this way requires TAP5000 to determine that PREG is not live past its use. In other words, the value from PREG is read once and not used again later. TAP5000 assumes by default that PREG is not live at exit points. If PREG is live at an exit point, you must specify this in the Translation Wizard dialog box. For more information on specifying register usage, see Section 2.5.2, *Setting Register Usage Information at Linkage Points*, page 2-10.

If the write and use instructions are in different basic blocks—that is, a branch (including the end of a repeat loop) or label intervenes between the write and the use—TAP5000 "hoists" the use into the same block as the write.

Some 'C5x instructions, such as MPYA, accumulate the previous product and produce the next one in parallel. On the 'C54x, a loop containing such an instruction is rotated to first produce a product and then accumulate it.

The table below provides a general outline of the translation in terms of the atomic write and use operations:

Define	**Use**	**'C54x translation**
MPY	PAC	MPY
MPY	ADD/SUB	MAC/MAS
MPY	SPL/SPH	MPY *tmp*, STH/STL *tmp*

The 'C5x has two instructions (SPL and SPH) to move PREG to memory, and one instruction (LPH) to load PREG from memory. Since PREG is generally a short-lived temporary register, these instructions may become unnecessary. However, TAP5000 avoids deleting memory references, and therefore may not be able to avoid translating these instructions.

In these cases, TAP5000 translates the instructions, but generates an efficiency hint for them (marked by a green square icon in the Source window). Double-clicking the icon opens the Translation Wizard dialog, which gives you to opportunity to omit these instructions.

4.2 Post-scaling Using PM Bits

This section corresponds to Section 3.2.3 of the applications document.

The ’C5x PREG shifter has 0, 1, 4 and -6 bit product shifts via the PM bits. The ’C54x has only 0 or 1-bit product shifts via FRCT. ’C54x does not have PM bits.

The PM mode bits in the ’C5x status register allow automatic post-scaling (shifting) of the multiplier output. The shift options are left-1, left-4, right-6, or none. The ’C54x has automatic post-scaling of left-1 via the FRCT bit, but cannot post-scale by left-4 or right-6. On the ’C54x, this type of scaling is either unnecessary (because of the guard bits) or accomplished in different ways (e.g., when storing the result).

TAP5000 models the PM left-1 shift using the ’C54x FRCT bit, but will not model the other PM shifts.

When PM is being written (via the SPM instruction), TAP5000 behaves as follows:

’C5x	Effect	’C54x translation
SPM 0	no shift	RSBX FRCT (accurately models behavior)
SPM 1	left-1	SSBX FRCT (accurately models behavior)
SPM 2	left-4	none (error generated)
SPM 3	right-6	none (error generated)

When PM is being used (by any instruction that shifts PREG), TAP5000 first tries to determine the value of PM. If PM is unknown, or known to be 2 or 3, TAP5000 issues a warning that the product will not be correctly scaled in translated code. In this case, you must modify the translated code to account for the scaling.

If your application does not use left-4 or right-6 post-scaling, it may be useful to globally disable the warnings related to PM:

1) From the Tools menu, select Translation Settings.
2) In the Translation dialog box, click the Other Options tab.
3) Remove the checkmark from the two warnings that reference PM.
4) Click OK.

4.3 Supplying the Value of DBMR

This section corresponds to Section 3.2.4 of the applications document.

The ’C5x DBMR register is used to hold bit masks for PLU instructions that perform direct logic operations on data memory. The ’C54x has no DBMR. Its instructions that perform logic operations on memory use only immediate operands.

TAP5000 tries to track the value of DBMR. If successful, it translates the use of DBMR to an immediate value, which is the most efficient translation. When register tracking fails, TAP5000 issues a warning, asking you to provide the DBMR value. Ultimately, if you cannot supply the value and it remains unknown, TAP5000 models DBMR as a memory location on the ’C54x. The DBMR memory location is loaded into an accumulator temporary register and the translation becomes a three-instruction read-modify-write sequence.

For example, consider the ’C5x instruction:

```
APL mem
```

The effect of this instruction is “*mem* &= DBMR”. If DBMR is known to contain a value of *val*, the translation is:

```
ANDM #val, mem
```

If the value of DBMR is unknown, the translation is:

```
LD    *(DBMR), acc
AND   mem, acc
STL   acc, mem
CMPM  mem, #0
```

4.4 Supplying the Value of BMAR

This section corresponds to Section 3.2.5 of the applications document.

The ’C5x BMAR register is used as an address register for block-memory instructions. The ’C54x has no BMAR register. Its block-memory instructions use either immediate address constants or indirect pointers in ARs.

TAP5000 tries to track the value of BMAR. When register tracking fails, TAP5000 issues a warning, asking you to provide the BMAR value. With a value for BMAR, it can translate the use of BMAR to an immediate address constant.

Ultimately, if the value is not known, TAP5000 models BMAR as a memory location on the ’C54x. The BMAR memory location is loaded into a temporary register, and the indirect form of the block-memory instruction is used. The exact translation sequence depends on the specific block-memory instruction being translated. The various cases are described below:

BLPD/BLDP — In this case, TAP5000 loads the value into accumulator A and uses READ1 or WRITEA to move between program and data memory.

BLDD — The ’C5x BLDD instruction allows a block move between a BMAR-addressed block and a direct-addressing mode. The ’C54x only supports indirect-to-indirect block moves. In this case, TAP5000 converts the direct operand to indirect by loading it (as well as BMAR) into an AR temporary register. TAP5000 can do this only when the direct address is specified symbolically.

MADS/MADD — In this case, TAP5000 loads the BMAR value into an AR and uses indirect addressing to access the coefficients. However, this access requires that the coefficients be moved from program memory to data memory. For more information, see Section 4.5, *Coefficients in Program Memory*, page 4-7.

Furthermore, if the MADS/MADD instruction is part of a repeat-single loop, the translation may not account for the auto-incrementing behavior on input data address. TAP5000 issues a warning for this problem.

4.5 Coefficients in Program Memory

Several of the ’C5x multiply-accumulate instructions perform two memory reads per cycle by using the program bus to read their second operand (typically a coefficient of some kind). Some, but not all, of the ’C54x multiply-accumulate instructions support program-memory reads, but only with constant addresses. These differences can cause the following translation problems:

- The ’C5x multiply-accumulate instructions that use program memory addressing (MAC, MACD, MADD, and MADS) can sometimes be translated into similar ’C54x instructions that use program memory addressing. When this type of translation is not possible, the instructions are translated into sequences that use data memory addressing. This translation requires that you manually move the coefficients from program to data memory. You may be able to do this in the linker.
- The MAC and MACD instructions use a program memory address specified as a 16-bit constant. When either of these instructions is used in a repeat-single loop, the program address is automatically incremented. But if the translation requires multiple instructions, the translation does not account for this auto-increment.

TAP5000 issues warnings to alert you if either (or both) of these conditions occur.

4.6 NDX Compatibility

This section corresponds to Section 3.2.6 of the applications document.

On the ’C5x, the INDX register is used for indexed addressing, and the ARCR register is used with the CMPR instruction for AR comparisons. The ’C54x has neither register. It overloads AR0 for both of these functions. The translation of operations involving INDX or ARCR depends on the setting of the compatibility mode (NDX) bit in the ’C5x status register.

In ’C2x compatibility mode (NDX == 0), any non-memory-mapped write to AR0 also writes INDX and ARCR. This write effectively preserves compatibility with the ’C2x, which, like the ’C54x, uses AR0 for all three functions. In this mode, any use of INDX or ARCR is translated to use AR0.

However, if the application writes to INDX and ARCR via their memory-mapped locations, they may contain different values than AR0. In this case, the translation would be invalid. In practice, there is no reason for an application to do this kind of write. Such a write would invalidate ’C2x compatibility.

In non-C2x-compatibility mode (NDX == 1), INDX and ARCR are modeled in memory locations on the ’C54x. Operations involving these registers are translated by loading the memory locations into AR0 and generating short sequences to mimic the operations. If AR0 is live, TAP5000 will insert additional code into the translated file to save and restore the register.

4.7 TRM Compatibility

This section corresponds to Section 3.2.7 of the applications document.

On the ’C5x, the TREG1 register is used for shift counts, and TREG2 is used for bit addresses in the BITT instruction. The ’C54x has neither register. It overloads T for both of these functions (shift counts can also use the ’C54x's ASM). The translation depends on the TRM compatibility bit. This bit allows ’C2x compatibility by tying TREG1 and TREG2 to the value of TREG0.

In ’C2x compatibility mode (TRM == 0), any non-memory-mapped write to TREG0 also writes TREG1 and TREG2. In this mode, any use of TREG1 or TREG2 is translated to use T.

However, if the application writes to TREG1 and TREG2 via their memory-mapped locations, they may contain different values than TREG0. In this case, the translation is invalid. In practice, there is no reason for an application to do this kind of write. Such a write would invalidate ’C2x compatibility.

In non-C2x-compatibility mode (TRM == 1), TREG1 and TREG2 are modeled in memory locations on the ’C54x. Operations involving these registers are translated by loading the memory locations into T and generating short sequences to mimic the operations. If T is live, TAP5000 will insert additional code into the translated file to save and restore the register.

4.8 Translation of Shift Instructions

The ’C5x has two shift instructions: SFL and SFR. Each of these can be translated in two possible ways:

- As an arithmetic shift (the default). In this case, the guard bits are preserved. For right shifts, the sign bit is propagated into the vacated bit positions.
- As a logical shift. In this case, the guard bits are cleared.

TAP5000 issues a warning when these instructions are translated and allows you to select the correct choice.

4.8.1 Effects of Logical/Arithmetic Shifts on Guard Bits

’C54x guard bits may be cleared by some logical operations.

The ’C5x has 32-bit accumulators. These accumulators are modeled by the ’C54x 40-bit accumulators, which have 8 extra, most-significant guard bits. Most translations are not affected by the presence of these extra bits because they interpret the value arithmetically. The value will be generated and interpreted the same way, but with more precision (more range).

However, when the accumulators are manipulated by logical instructions such as bit-wise ANDs, ORs, XORs, rotates, or shifts, the extra bits may cause the results to differ after translation.

In particular, for some shift or rotate operations, the translation results in the guard bits being cleared. For other shift and rotate operations, the translation results in the guard bits being sign-extended from the register value. Depending on how the result will be interpreted, the translated behavior may or may not be desirable. TAP5000 issues warnings to alert you of possible problems.

Individual cases are summarized as follows:

- The translation of single-register shifts using arithmetic shifts preserves the guard bits.
- The translation of rotate instructions using rotate instructions clears the guard bits.
- The translation of dual-register shifts and rotates using logical shifts and rotates clears the guard bits.
- If the guard bits are not live, the translation is always accurate. If they are live, the translation may or may not be accurate.

4.9 Overflow Behavior

This section corresponds to Section 3.2.10 of the applications document.

In some cases, the translated code can produce an overflow condition when the source code will not:

- In the translation of instructions that read from memory with a 16-bit left shift, when SXM is 0. For example:

```
LACC mem, 16
ADD mem, 16
SUB mem, 16
ZALR  mem
```

- In translations that generate arithmetic left shifts (SFTA). The overflow occurs in the translation of the 'C5x SFL (left shift) instructions when translated as an arithmetic shift, or in the translation of the SFLB (left shift through B) instruction.

If overflow does occur in these situations, it can potentially change the behavior of the code in two ways:

- The OV bit will be set. If the code is not expecting this instruction to set OV, the translated code will not run correctly.
- If OVM (overflow mode) is 1 when overflow occurs, the result saturates. This saturation can cause a different value to be computed.

Note: If OVM is 1 when overflow occurs, you should clear the OVM bit.

For best results, if OVM is 1 when overflow occurs, you should explicitly clear the OVM bit in the translated code, unless you need saturation.

In any of these cases, TAP5000 issues a warning to alert you of the potential problem.

4.10 Supplying the Value of ARP

This section corresponds to Section 3.2.11 of the applications document.

The ’C5x uses the ARP field, as set by a previous instruction, to specify the AR to be used for indirect addressing. The ’C54x specifies the AR directly in the instruction itself. To provide a correct translation, TAP5000 needs to know the value of ARP at each instance of indirect addressing.

TAP5000 uses register tracking to determine the value of ARP. Tracking is usually powerful enough to correctly track the value through control flow discontinuities (branches and labels), unless calls and externally-accessible labels force the value to be flushed. Note that TAP5000 cannot track the ARP across calls unless you specify (at the call site or via the Calling Conventions dialog) that it is preserved.

4.10.1 Supplying the Value of ARP for Entry Points

You can specify the ARP at any entry point, or as a default global value for all entry points. Setting the ARP globally (in the Calling Conventions dialog) is useful for ’C5x applications that use a convention to define the ARP value at entry and exit points.

For example, suppose the ’C5x application uses AR1 as a software stack pointer. By convention, the ARP points to AR1 at the entry and exit from every routine. You can specify this information in the Calling Conventions dialog:

1) From the Tools menu, select Calling Conventions.
2) In the Calling Conventions dialog, click the Calls (Inputs) tab. On this page, select AR1 and ARP. (They are selected when a checkmark appears in the box next to their names.)
3) Click the Calls (Outputs) tab. On this page, select AR1 and ARP.
4) Click the Calls (Preserved) tab. On this page, select AR1 and ARP.
5) Click the Default ARP page. On this page, select AR1.
6) Click OK.

You can override these settings locally at any entry point where the convention does not apply. For more information on setting registers locally, see Section 2.5.2, *Setting Register Usage Information at Linkage Points*, page 2-10.

Note that the default ARP value supplied in the Calling Conventions dialog does not apply to any point at which the ARP is unknown. The value only applies at entry points.

4.11 Translations Affecting Status Bits

This section corresponds to Section 3.2.14 of the applications document.

Most translations affect, and are affected by, status bits in ways that are similar to the original code. If a translation may affect the status bits differently than the original code did, TAP5000 handles the situation in one of three ways:

- ❏ TAP5000 uses its register usage analysis to detect whether the difference is significant. If the code is unaffected by the difference (for example, no subsequent instruction uses the affected bit), TAP5000 quietly ignores the issue.
- ❏ If TAP5000 can generate a single instruction as part of the translation that causes the status to be correctly set, it will do so. An efficiency hint icon (a green square) will appear in the Source window at the appropriate source line. Use the Translation Wizard to remove the extra instruction if you can determine the difference is actually immaterial.
- ❏ If more than one instruction would be required to set the status in the same way, TAP5000 ignores the problem and issues a warning (a yellow diamond icon in the Source window). If you can determine the difference is immaterial, you can suppress the warning.

For more information on responding to translation issues, see Section 2.5.1, *Responding to a Translation Issue Via the Translation Wizard*, page 2-9.

4.12 Condition Combinations

This section corresponds to Section 3.2.15 of the applications document.

The ’C5x and ’C54x both allow multiple conditions to be specified on conditional instructions (such as branches, calls, returns, and XC). However, not all combinations allowed by the ’C5x are allowed on the ’C54x.

In these cases, TAP5000 issues an error.

4.13 Translation of Interrupts

This section discusses the various translation issues related to interrupts.

4.13.1 Interrupt Vector Numbers

This section corresponds to Section 3.2.16 of the applications document.

Interrupts are numbered differently on the ’C5x and ’C54x. In the translation of the INTR instruction, TAP5000 maps the ’C5x interrupt numbers to the corresponding ’C54x numbers. However, this mapping may not be correct in all cases. TAP5000 issues a warning message so that you can adjust the numbers if necessary.

However, TAP5000 does not rearrange the interrupt vector table according to the ’C54x interrupt numbers. The table must be updated by hand.

4.13.2 No Automatic Context Save and Restore for Interrupts

This section corresponds to Section 3.2.17 of the applications document.

The ’C5x uses its shadow registers for automatic context save and restore. The ’C54x has no automatic context save mechanism. Interrupt service routines (ISRs) must explicitly save and restore context using pushes and pops to the software stack.

TAP5000 has no provision for detecting ISRs and cannot generate the necessary context save and restore code.

When writing this code, you should be sure to include, as part of the context, any of the memory locations being used to model the following ’C5x registers: ARCR, BMAR, CBER1, CBER2, CBSR1, CBSR2, DBMR, INDX, PM, TREG1, and TREG2.

In general, TAP5000 cannot detect which entry points correspond to ISRs. To alert you of this potential issue, TAP5000 issues warnings in two cases:

- ❏ when translating a return-from-interrupt instruction
- ❏ when translating an INTR (software interrupt) instruction

4.13.3 Interrupts and Dedicated Register Usage

If a register is expected to contain some value when an interrupt occurs, or if the register is changed by an interrupt routine without being saved and restored, the register should be specified as dedicated in the Calling Conventions dialog box.

Marking the register as dedicated ensures that TAP5000 correctly maintains the register, and avoids using it as a temporary register.

For more information on dedicated registers, see Section 3.5.3, *Dedicated Registers*, page 3-10. For more information on the Calling Conventions dialog, see Section 2.7, *Setting Global Calling Convention Defaults*, page 2-13.

4.14 Status Bit Positions

This section corresponds to Section 3.2.20 of the applications document.

Some status bits common to the ’C5x and ’C54x are in different status registers or in different positions within the same register. This issue affects code that manipulates bits in these registers directly (for example, by copying a status register into memory, setting or reading bits in the memory location, or copying back to the status register).

TAP5000 does not try to detect or correctly translate this behavior. It alerts you to the potential problem on instructions that move between status registers and memory (such as LST and SST).

4.15 Translation of XC Instructions

Both the ’C5x and ’C54x have conditional execute (XC) instructions. However, three issues can arise in the translation of the XC instruction:

- The ’C54x requires two cycles of latency between the generation of the condition and its use by XC. TAP5000 correctly schedules the translation to account for this.

  However, the TAP5000 also requires that the condition remain unchanged during these 2 cycles. TAP5000 does not account for this requirement, and issues a warning if it detects a possible violation. In the case of a violation, you must edit the translated code manually.

- TAP5000 cannot translate XC instructions that occur in the delay slot of a branch or call. It attempts to reorder the XC out of the branch and rewrite it as a conditional branch.

  If TAP5000 cannot find a way to rewrite the XC, it issues an error message. In this case, you must edit the source code to remove the XC from the delay slot.

- XC allows up to two instructions to be executed conditionally. If the translation requires more than two instructions, TAP5000 rewrites the XC using branching instructions.

4.16 Repeat Loops

The ’C5x and ’C54x have similar hardware-looping (repeat) mechanisms. TAP5000 can usually translate a single-instruction ’C5x repeat loop to a single-instruction ’C54x repeat loop. However, the repeated instruction may translate to more than one instruction. In this case, TAP5000 rewrites the single-repeat loop as a BANZ loop, and allocates an AR as a temporary register to act as the loop counter.

TAP5000 may also deal with a repeat-single loop in the following ways:

- The MAC and MACD instructions use a program memory address specified as a 16-bit constant. When either of these instructions is used in a repeat-single loop, the program address is automatically incremented. But if the translation requires multiple instructions, the translation does not account for this auto-increment.
- If a MADS/MADD instruction is part of a repeat-single loop, the translation may not account for the auto-incrementing behavior on input data address. TAP5000 issues a warning for this problem.

4.17 Memory-Mapped Register Addressing

Both the ’C5x and ’C54x include registers that can be addressed as memory locations. However, the ’C5x and ’C54x have different sets of registers, and some of the registers in common are located at different addresses. TAP5000 tries to detect accesses to MMRs and translate them to the corresponding ’C54x register when possible.

However, certain translation issues may arise, depending on how the MMR is accessed:

- Several instructions use dedicated MMR addressing (LAMM, SAMM, LMMR, SMMR). If the operand is a named register or a constant, TAP5000 knows which MMR is being referenced and translates it when possible. But if the operand is indirect, two issues arise:
  - TAP5000 cannot translate the MMR reference because it does not know what register is being accessed. The reference will be to the same memory location on the ’C54x as the ’C5x, but it may not be the same register. TAP5000 issues a warning in this case.
  - The ’C54x translation of these instructions causes the upper bits of the AR to be cleared. If the AR is live following the access, TAP5000 accounts for this difference by saving and restoring the AR. For more information on register liveness, see Section 3.5, *Analysis of Register Usage*, page 3-8.
- On the ’C5x, MMRs are commonly accessed by setting DP to 0 and using direct addressing. If TAP5000 can detect that DP is 0, it determines that such references actually address MMRs and translates them accordingly. However, if TAP5000 cannot detect that DP is 0 in a direct addressing situation, it cannot translate the address. Likewise, if an MMR is addressed using indirect addressing, TAP5000 cannot determine which register is being addressed and cannot translate its address.
- It is also possible to use MMR names as immediate constants in the program. For example, the ’C5x instruction "LACC #INDX" loads the memory-mapped address of the INDX register into the accumulator. Since TAP5000 cannot determine the context in which the constant is used, it does not translate such addresses to their ’C54x equivalents.

4.18 Circular Addressing

The mechanisms for circular addressing on the ’C5x are substantially different than those on the ’C54x. The ’C5x uses dedicated control registers that modify the regular indirect addressing modes to operate in circular fashion. The ’C54x has special addressing modes.

TAP5000 cannot detect when circular addressing is being used, and does not attempt to translate it. The tool issues a warning message when it detects accesses to the ’C5x circular addressing registers.

4.19 Peripherals and I/O

Although the ’C5x and ’C54x families have similar peripherals, the specific peripherals vary from device to device. Generally, there is not a direct mapping from one to the other. Peripherals are usually accessed via memory-mapped registers that control them.

TAP5000 does not try to translate MMRs that control peripherals. Instead, it issues warnings for these MMRs and translates them to memory references that have the same name as the original C5x MMR. If the ’C54x target device has a register with the same name, the translated code is probably correct without modification. If not, the translated code will not assemble and will have to be manually rewritten to use a different peripheral.

Similarly, the ’C54x does not support memory-mapped access to I/O ports. Code that uses this ’C5x feature will not translate.

4.20 Handling Macros

Macro definitions cannot be translated. The translation of a macro definition may be dependent on its invocation sites. Because TAP5000 translates one source file at a time, it cannot analyze all possible invocation sites to see how the macro is used. TAP5000 preserves macro definitions as comments.

However, TAP5000 does translate expanded macro invocations. The tool inserts the macro expansion code as comments after the actual invocation in the Source window. It translates the macro expansion code, which then appears in the Translation window. The macro invocation is preserved as a comment in the Translation window above the first instruction of the translated expansion.

You must edit the translated output file by hand to re-insert the macro definitions and invocations.

4.21 Conditional Assembly

The translation of conditional assembly is limited. TAP5000 translates only one configuration of the conditional source: the "true" configuration. The condition is evaluated to be “true”, and the corresponding block of code for the “true” condition is translated. The code for the “false” condition is not translated. You must translate the "false" block of conditional code by hand.

The code that follows the conditional assembly may be dependent on code contained in the conditional assembly. You will need to check the translation to ensure that these dependencies are properly translated.

4.22 Stack Management

The ’C5x has a seven-location hardware stack. The ’C54x has no hardware stack but has a dedicated stack pointer for a software stack anywhere in data memory. TAP5000 can model the ’C5x hardware stack using the ’C54x software stack, assuming that the code does not depend on the limited depth of the stack.

If your ’C5x code adds a software stack using an AR as a stack pointer, you should modify the translated code to use the ’C54x stack and dedicated stack pointer instead of the AR.

4.23 Translation of Directives

There are various issues related to the translation of certain ’C5x assembler directives. These issues are described below:

- **Redefinition of reserved symbols (.set, .equ).** TAP5000 will not translate a .set or .equ directive in the source code that defines a symbol that corresponds to a ’C54x register name or other predefined symbol. A warning will be issued.

  In addition, TAP5000 does not detect or correct user-defined symbol names that conflict with ’C54x reserved names. User-defined symbol names may also conflict with translator-defined symbols. TAP5000 is sometimes able to detect this, but not always. If a naming conflict is not detected, the translated code may not assemble.

- **.string directive.** The .string directive behaves differently on the ’C5x than it does on the ’C54x. On the ’C5x, the value operands are packed as two bytes per word. On the ’C54x, each byte of data is encoded into a word. The ’C5x .string directive will be translated to a ’C54x .pstring directive.

- **Data blocking (.blong, .bfloat).** The ’C5x provides some directives that imply "blocking" of the data. In this case, blocking means that the value will not span a page boundary. The ’C54x does not support blocking for initialized data directives. These directives will simply be translated into their non-blocking equivalents (.long and .float).

- **Conditional assembly directives (.if, .elseif, .else, .endif).** Code contained inside false conditional blocks will not be translated. This code will be included in the translated file (and thus, in the Translation window) as comments. The conditional directives themselves will be translated in tact.

- **Absolute section directive (.asect).** The .asect directive on the ’C5x will be translated into an initialized section directive on the ’C54x. The .asect directive is not supported on the ’C54x.

- **Word ordering for 32-bit initialized data directives (.long, .float).** The ordering of these directives is different for ’C5x (least significant word store first) and ’C54x (most significant word stored first). The translation of code that accesses .long or .float data does not take these differences into account. You may need to restructure code that references the data defined by these directives.

Appendix A

Glossary

A

absolute address: An address that is permanently assigned to a TMS320C54x memory location.

allocation: A process in which the linker calculates the final memory addresses of output sections.

ASCII: American Standard Code for Information Interchange. A standard computer code for representing and exchanging alphanumeric information.

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

byte: A sequence of eight adjacent bits operated upon as a unit.

C

call site: A point at which control exits the function and returns at the same point. For example, a call, a trap, or a software interrupt.

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

comment: A source statement (or portion of a source statement) that is used to document or improve readability of a source file. Comments are not compiled, assembled, or linked; they have no effect on the object file.

common object file format (COFF): A binary object file format that promotes modular programming by supporting the concept of *sections*. All COFF sections are independently relocatable in memory space; you can place any section into any allocated block of target memory.

configured memory: Memory that the linker has specified for allocation.

constant: A numeric value that does not change and that can be used as an operand.

D

dead register: A register is considered live if it currently contains a value that will be used subsequently by other instructions. If it does not, the register is considered to be dead.

directives: Special-purpose commands that control the actions and functions of a software tool (as opposed to assembly language instructions, which control the actions of a device).

E

emulator: A hardware development system that emulates TMS320C54x operation.

entry point: any point at which control may enter a given function. For example, the first instruction in a function, any globally visible (via .global or .def) labels defined in the function, labelled instructions that can be reached via indirect branches or calls, or regions of code that appear to be otherwise unreachable.

executable module: An object file that has been linked and can be executed in a TMS320C54x system.

exit point: any point at which control may exit a given function. For example, the last instruction in a function, branches to other functions or externally visible (via a .global or .ref) labels, indirect branches, or return instructions.

expression: One or more operations in assembler programming represented by a combination of symbols, constants, and paired parentheses separated by arithmetic operators.

F

function: TAP5000 separates the code in a source file into disjoint, logical units that connect at interface points called linkage points. These units are referred to as functions.

G

global symbol: 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.

L

label: A symbol that begins in column 1 of a source statement and corresponds to the address of that statement.

linkage point: Certain points in a program where control transfers from one function to another. Linkage points affect the translation process.

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.

liveness: A register is considered *live* if it currently contains a value that will be used subsequently by another instructions. If it does not, it's considered *dead*.

loader: A device that loads an executable module into TMS320C54x system memory.

M

macro: A user-defined routine that can be used as an instruction.

macro call: The process of invoking a macro.

macro definition: A block of source statements that define the name and the code that make up a macro.

macro expansion: The source statements that are substituted for the macro call and are subsequently assembled.

macro invocation: The process of invoking a macro; it is known as a macro call.

macro library: An archive library composed of macros. Each file in the library must contain one macro; its name must be the same as the macro name it defines, and it must have an extension of .asm.

map file: An output file, created by the linker, that shows the memory configuration, section composition, and section allocation, as well as symbols and the addresses at which they were defined.

memory map: A map of target system memory space, which is partitioned into functional blocks.

mnemonic: An instruction name that the assembler translates into machine code.

model statement: Instructions or assembler directives in a macro definition that are assembled each time a macro is invoked.

O

object file: A file that has been assembled or linked and contains machine-language object code.

object library: An archive library made up of individual object files.

operands: The arguments, or parameters, of an assembly language instruction, assembler directive, or macro directive.

options: Command parameters that allow you to request additional or specific functions when you invoke a software tool.

output module: A linked, executable object file that can be downloaded and executed on a target system.

S

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

simulator: A software development system that simulates TMS320C54x operation.

source file: A file that contains assembly language code that will be assembled to form an object file.

symbol: A string of alphanumeric characters that represents an address or a value.

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

temporary register: A register that is used temporarily by TAP5000.

translation assistant program: The TAP5000 translation assistant program converts assembly code written in the 'C5x instruction set to code written in the ’C54x instruction set.

W

well-defined expression: A term or group of terms that contains only symbols or assembly-time constants that have been defined before they appear in the expression.

word: A 16-bit addressable location in target memory.

Index

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z