

TriCore v1.5

C CROSS-COMPILER USER'S GUIDE

A publication of
Altium BV
Documentation Department
Copyright © 2002 Altium BV

All rights reserved. Reproduction in whole or part is prohibited
without the written consent of the copyright owner.

TASKING is a brand name of Altium Limited.

The following trademarks are acknowledged:

FLEXlm is a registered trademark of Globetrotter Software, Inc.

HP and HP-UX are trademarks of Hewlett-Packard Co.

Intel is a trademark of Intel Corporation.

Motorola is a registered trademark of Motorola, Inc.

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

SUN is a trademark of Sun Microsystems, Inc.

UNIX is a registered trademark of X/Open Company, Ltd.

All other trademarks are property of their respective owners.

Data subject to alteration without notice.

<http://www.tasking.com>
<http://www.altium.com>

The information in this document has been carefully reviewed and is believed to be accurate and reliable. However, Altium assumes no liabilities for inaccuracies in this document. Furthermore, the delivery of this information does not convey to the recipient any license to use or copy the software or documentation, except as provided in an executed license agreement covering the software and documentation.

Altium reserves the right to change specifications embodied in this document without prior notice.

CONTENTS

TABLE OF CONTENTS



TASKING



CONTENTS

SOFTWARE INSTALLATION **1-1**

1.1	Introduction	1-3
1.2	Installation for Windows	1-3
1.2.1	Setting the Environment	1-4
1.3	Installation for Linux	1-5
1.3.1	RPM Installation	1-5
1.3.2	Tar.gz Installation	1-6
1.3.3	Setting the Environment	1-7
1.4	Installation for UNIX Hosts	1-8
1.4.1	Setting the Environment	1-10
1.5	Licensing TASKING Products	1-11
1.5.1	Obtaining License Information	1-11
1.5.2	Installing Node-Locked Licenses	1-12
1.5.3	Installing Floating Licenses	1-13
1.5.4	Starting the License Daemon	1-15
1.5.5	Setting Up the License Daemon to Run Automatically	1-16
1.5.6	Modifying the License File Location	1-17
1.5.7	How to Determine the Hostid	1-19
1.5.8	How to Determine the Hostname	1-19

OVERVIEW **2-1**

2.1	Introduction to TriCore C Cross-Compiler	2-3
2.2	Product Definition	2-4
2.3	General Implementation	2-5
2.3.1	Compiler Phases	2-5
2.3.2	Frontend Optimizations	2-6
2.3.3	Backend Optimizations	2-9
2.4	Compiler Structure	2-10
2.5	Environment Variables	2-13
2.6	Sample Session	2-14
2.6.1	Using EDE	2-14
2.6.2	Using the Control Program	2-21
2.6.3	Using the Makefile	2-23

LANGUAGE IMPLEMENTATION		3-1
3.1	Introduction	3-3
3.2	Accessing Memory	3-4
3.2.1	Storage Types	3-4
3.2.2	The <code>_at()</code> Attribute	3-6
3.2.3	The <code>_atbit()</code> Attribute	3-7
3.3	Data Types	3-8
3.3.1	Signed Characters	3-9
3.3.2	ANSI C Type Conversions	3-9
3.4	Fractional Data Types	3-13
3.4.1	Additional Basic Types	3-13
3.4.2	Type Conversions	3-13
3.4.3	Promotion Rules	3-14
3.4.4	Intrinsic Functions	3-15
3.5	Type Qualifier <code>_sat</code>	3-16
3.6	Packed Data Types	3-17
3.6.1	Additional Basic types	3-17
3.6.2	Intrinsic Functions	3-17
3.6.3	Halfword Packed Unions and Structures	3-19
3.7	Bit Data Types	3-20
3.7.1	The <code>_bit</code> Type	3-20
3.7.2	Type Qualifier <code>_sfrbit16</code> and <code>_sfrbit32</code>	3-21
3.8	Parameter Passing	3-23
3.9	Function Qualifiers	3-24
3.9.1	Interrupt Function Qualifier	3-24
3.9.2	Trap Function Qualifiers	3-25
3.9.3	Enable Interrupt/Trap Function Qualifier	3-27
3.9.4	BISR Interrupt/Trap Function Qualifier	3-28
3.9.5	System Call Function Qualifier	3-29
3.9.6	Stack Model Function Qualifier	3-29
3.9.7	Far Function Storage Qualifier	3-30
3.10	Type Qualifier <code>volatile</code>	3-32
3.11	Type Qualifiers <code>restrict</code> and <code>_restrict</code>	3-33
3.12	Strings	3-34

3.13	Variable Argument Lists	3-35
3.14	Inline C Functions	3-36
3.15	Inline Assembly	3-37
3.16	Intrinsic Functions	3-40
3.17	MISRA C	3-44
3.18	Structure Tags	3-45
3.19	Typedef	3-45
3.20	Circular Buffers	3-46
3.21	Switch Statement	3-47

COMPILER USE **4-1**

4.1	Control Program	4-3
4.2	Compiler	4-6
4.2.1	Detailed Description of the Compiler Options	4-10
4.3	Include Files	4-68
4.4	Pragmas	4-71
4.5	Compiler Limits	4-78

COMPILER DIAGNOSTICS **5-1**

5.1	Introduction	5-3
5.2	Return Values	5-4
5.3	Errors and Warnings	5-5

LIBRARIES **6-1**

6.1	Introduction	6-3
6.2	Header Files	6-3
6.3	C Libraries	6-4
6.3.1	Single Precision Floating Point	6-6
6.3.2	C Library Implementation Details	6-6
6.3.3	C Library Interface Description	6-12
6.3.4	C Library Reentrancy	6-62
6.4	Run-time Library	6-72

RUN-TIME ENVIRONMENT **7-1**

7.1	Startup Code	7-3
7.2	Register Usage	7-7
7.3	Stack	7-9
7.4	Heap	7-10
7.5	Floating Point Arithmetic	7-10
7.5.1	Compliance to IEEE-754	7-11
7.5.2	Special Floating Point Values	7-13
7.5.3	Trapping Floating Point Exceptions	7-13
7.5.4	Floating Point Trap Handling API	7-15

FLEXIBLE LICENSE MANAGER (FLEXlm) **A-1**

1	Introduction	A-3
2	License Administration	A-3
2.1	Overview	A-3
2.2	Providing For Uninterrupted FLEXlm Operation	A-5
2.3	Daemon Options File	A-7
3	License Administration Tools	A-8
3.1	lmcksum	A-10
3.2	lmdia (Windows only)	A-11
3.3	lmdown	A-12
3.4	lmgrd	A-13
3.5	lmhostid	A-15
3.6	lmremove	A-16
3.7	lmreread	A-17
3.8	lmstat	A-18
3.9	lmswchr (Windows only)	A-20
3.10	lmver	A-21
3.11	License Administration Tools for Windows	A-22
3.11.1	LMTOOLS for Windows	A-22
3.11.2	FLEXlm License Manager for Windows	A-23

4	The Daemon Log File	A-25
4.1	Informational Messages	A-26
4.2	Configuration Problem Messages	A-29
4.3	Daemon Software Error Messages	A-31
5	FLEXlm License Errors	A-33
6	Frequently Asked Questions (FAQs)	A-37
6.1	License File Questions	A-37
6.2	FLEXlm Version	A-37
6.3	Windows Questions	A-38
6.4	TASKING Questions	A-39
6.5	Using FLEXlm for Floating Licenses	A-41

MISRA C

B-1

CPU FUNCTIONAL PROBLEMS

C-1

1	Introduction	C-3
2	CPU Functional Problem bypasses TC1 V1.2	C-5
2.1	TC112_COR1	C-5
2.2	TC112_COR3	C-6
2.3	TC112_COR4	C-7
2.4	TC112_COR6	C-8
2.5	TC112_COR10	C-9
2.6	TC112_COR13	C-10
2.7	TC112_COR14	C-11
2.8	TC112_COR15	C-13
2.9	TC112_COR16	C-14
2.10	TC112_COR17	C-15
3	CPU Functional Problem bypasses TC1 V1.3	C-16
3.1	TC113_PMU1	C-16
3.2	TC113_PMU3	C-17
3.3	TC113_CPU5	C-18
3.4	TC113_CPU9	C-19
3.5	TC113_CPU11	C-20



3.6	TC113_CPU13	C-21
3.7	TC113_CPU14	C-22
3.8	TC113_CPU15	C-23
3.9	TC113_CPU16	C-24
3.10	TC113_DMU1	C-25
3.11	TC113_LFI2	C-26
3.12	TC113_LFI3	C-27

INDEX

MANUAL PURPOSE AND STRUCTURE

PURPOSE

This manual is aimed at users of the TASKING TriCore C Cross-Compiler. It assumes that you are familiar with the C language.

MANUAL STRUCTURE

Related Publications

Conventions Used In This Manual

1. Software Installation
Describes the installation of the C Cross-Compiler for the TriCore family of processors.
2. Overview
Provides an overview of the TASKING TriCore toolchain and gives you some familiarity with the different parts of it and their relationship. A sample session explains how to build a TriCore application from your C file.
3. Language Implementation
Concentrates on the approach of the TriCore architecture and describes the language implementation. The C language itself is not described in this document. We recommend: "The C Programming Language" (second edition) by B. Kernighan and D. Ritchie (1988, Prentice Hall).
4. Compiler Use
Deals with control program and C compiler invocation, command line options and pragmas.
5. Compiler Diagnostics
Describes the exit status and error/warning messages of the compilers.
6. Libraries
Contains the library functions supported by the compilers and describes their interface and 'header' files.

7. Run-time Environment

Describes the run-time environment for a C application. It deals with items like assembly language interfacing, C startup code and stack/heap size.

APPENDICES

A. Flexible License Manager (FLEXlm)

Contains a description of the Flexible License Manager.

B. MISRA C

Supported and unsupported MISRA C rules.

C. CPU Functional Problems

Describes how the TriCore toolchain can bypass some functional problems of the CPU.

INDEX

RELATED PUBLICATIONS

- The C Programming Language (second edition) by B. Kernighan and D. Ritchie (1988, Prentice Hall)
- ANSI X3.159-1989 standard [ANSI]
- ISO/IEC 9899:1999(E), Programming languages – C [ISO/IEC]
More information on the standards can be found at
<http://www.ansi.org>
- TriCore C Cross-Assembler, Linker/Locator, Utilities User's Guide [TASKING, MA060-000-00-00]
- TriCore CrossView Pro Debugger User's Guide [TASKING, MA060-043-00-00]
- TriCore Architecture Manual [1999, Infineon]
- TriCore Architecture v1.3 Manual [2000, Infineon]

CONVENTIONS USED IN THIS MANUAL

The notation used to describe the format of call lines is given below:

{ }	Items shown inside curly braces enclose a list from which you must choose an item.
[]	Items shown inside square brackets enclose items that are optional.
	The vertical bar separates items in a list. It can be read as OR.
<i>italics</i>	Items shown in italic letters mean that you have to substitute the item. If italic items are inside square brackets, they are optional. For example: <i>filename</i> means: type the name of your file in place of the word <i>filename</i> .
...	An ellipsis indicates that you can repeat the preceding item zero or more times.
screen font	Represents input examples and screen output examples.
bold font	Represents a command name, an option or a complete command line which you can enter.

For example

command [*option*]... *filename*

This line could be written in plain English as: execute the command *command* with the optional options *option* and with the file *filename*.

Illustrations

The following illustrations are used in this manual:



This is a note. It gives you extra information.



This is a warning. Read the information carefully.



This illustration indicates actions you can perform with the mouse.



This illustration indicates keyboard input.



This illustration can be read as “See also”. It contains a reference to another command, option or section.



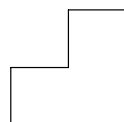
CHAPTER

1

SOFTWARE INSTALLATION



TASKING



1

CHAPTER

1.1 INTRODUCTION

This chapter describes how you can install the TASKING C Cross-Compiler for the TriCore on Windows 95/98/NT/2000, Linux and several UNIX hosts.

1.2 INSTALLATION FOR WINDOWS

Step 1

Start Windows 95/98/NT/2000, if you have not already done so.

Step 2

Insert the CD-ROM into the CD-ROM drive.

If the TASKING Welcome dialog box appears, skip to Step 5. Otherwise, continue from Step 3.

Step 3

Select the Start button and select the Run . . . menu item.

Step 4

On the command line type:

d:\setup

(substitute the correct drive letter for your CD-ROM drive) and press the **<Return>** or **<Enter>** key or click on the OK button.

The TASKING Welcome dialog box appears.

Step 5

Select a product and click on Install.

Step 6

Follow the instructions that appear on your screen.



You can find your serial number on the *Certificate of Authenticity* or *Product Update Form*, delivered with the product.

Step 7

License the software product as explained in section 1.5, *Licensing TASKING Products*.

1.2.1 SETTING THE ENVIRONMENT

After you have installed the software, you can set some environment variables to make invocation of the tools easier, when invoking the tools from a command prompt. A list of all environment variables used by the tool chain is present in the section *Environment Variables* in the chapter *Overview*.

Make sure that your path is set to include all of the executables you have just installed. If you installed the software under C:\CTRI, you can include the executable directory C:\CTRI\BIN in your search path.

The environment variable TMPDIR can be used to specify a directory where programs can place temporary files. The compiler uses the environment variable CTRIINC to search for include files. An example of setting this variable is given below.



See also the section *Include Files* in the chapter *Compiler Use*.

Example Windows Command Prompt

Enter the following line when you use a Command Prompt window.

```
set CTRIINC=c:\ctri\include
```

Example Windows 95/98

Add the following line to your autoexec.bat file.

```
set CTRIINC=c:\ctri\include
```

Example Windows NT / 2000

1. Open the System Properties dialog.

You can do this by double-clicking on the System icon in the Control Panel (Start | Settings | Control Panel) or right-click on the My Computer icon on your desktop and select Properties.

2. Select the Environment tab.

3. In the Variable edit field enter:

CTRIINC

4. In the Value edit field enter:

c:\ctri\include

5. Click on the Set button, then click OK.

1.3 INSTALLATION FOR LINUX

Each product on the CD-ROM is available as an RPM package and as a gzipped tar file. For each product, the following files are present:

```
SWproduct-version-RPMrelease.i386.rpm  
SWproduct-version.tar.gz
```

Both files contain exactly the same information. When your Linux distribution supports RPM packages, you can install the .rpm file. Otherwise, you can install the product from the .tar.gz file.

1.3.1 RPM INSTALLATION

Step 1

In most situations you have to be "root" to install RPM packages, so either login as "root", or use the **su** command.

Step 2

Insert the CD-ROM into the CD-ROM drive. Mount the CD-ROM on a directory, for example /cdrom. See the Linux manual pages about **mount** for details.

Step 3

Go to the directory on which the CD-ROM is mounted:

```
cd /cdrom
```

Step 4

To install or upgrade all products at once, issue the following command:

```
rpm -U SW*.rpm
```

This will install or upgrade all products in the default installation directory `/usr/local`. Every RPM package will create a single directory in the installation directory.

The RPM packages are 'relocatable', so it is possible to select a different installation directory with the **--prefix** option. For instance when you want to install the products in `/opt`, use the following command:

```
rpm -U --prefix /opt SW*.rpm
```



For Red Hat 6.0 users: The **--prefix** option does not work with RPM version 3.0, included in the Red Hat 6.0 distribution. Please upgrade to RPM version 3.0.3 or higher, or use the `.tar.gz` file installation described in the next section if you want to install in a non-standard directory.

1.3.2 TAR.GZ INSTALLATION

Step 1

Login as a user.

Be sure you have read, write and execute permissions in the installation directory. Otherwise, login as "root" or use the **su** command.

Step 2

Insert the CD-ROM into the CD-ROM drive. Mount the CD-ROM on a directory, for example `/cdrom`. See the Linux manual pages about **mount** for details.

Step 3

Go to the directory on which the CD-ROM is mounted:

```
cd /cdrom
```

Step 4

To install the products from the `.tar.gz` files in the directory `/usr/local`, issue the following command for each product:

```
tar xzf SWproduct-version.tar.gz -C /usr/local
```

Every `.tar.gz` file creates a single directory in the directory where it is extracted.

1.3.3 SETTING THE ENVIRONMENT

After you have installed the software, you can set some environment variables to make invocation of the tools easier. A list of all environment variables used by the toolchain is present in the section *Environment Variables* in the chapter *Overview*.

Make sure that your path is set to include all of the executables you have just installed.

The environment variable `TMPDIR` can be used to specify a directory where programs can place temporary files.

1.4 INSTALLATION FOR UNIX HOSTS

Step 1

Login as a user.

Be sure you have read, write and execute permissions in the installation directory. Otherwise, login as root or use the **su** command.

Step 2

If you are a first time user decide where you want to install the product (By default it will be installed in `/usr/local`).

Step 3

For CD-ROM install: insert the CD-ROM into the CD-ROM drive. Mount the CD-ROM on a directory, for example `/cdrom`. Be sure to use a ISO 9660 file system with Rock Ridge extensions enabled. See the UNIX manuals page about **mount** for details.

Or:

For tape install: insert the tape into the tape unit and create a directory where the contents of the tape can be copied to. Consider the created directory as a temporary workspace that can be deleted after installation has succeeded. For example:

```
mkdir /tmp/instdir
```

Step 4

For CD-ROM install: go to the directory on which the CD-ROM is mounted:

```
cd /cdrom
```

For tape install: copy the contents of the tape to the temporary workspace using the following commands:

```
cd /tmp/instdir
tar xvf /dev/tape
```

where *tape* is the name of your tape device.



If you have received a tape with more than one product, use the non-rewinding device for installing the products.

Step 5

Run the installation script:

```
sh install
```

and follow the instructions appearing on your screen.

First a question appears about where to install the software. The default answer is **/usr/local**. On certain sites you may want to select another location.

On some hosts the installation script asks if you want to install SW000098, the Flexible License Manager (FLEXlm). If you do not already have FLEXlm on your system, you must install it; otherwise the product will not work on those hosts. See section 1.5, *Licensing TASKING Products*.

If the script detects that the software has been installed before, the following messages appear on the screen:

```
*** WARNING ***  
SWxxxxxxx xxxx.xxxx already installed.  
Do you want to REINSTALL? [y,n]
```

Answering **n** (no) to this question causes installation to abort and the following message being displayed:

```
=> Installation stopped on user request <=
```

Answering **y** (yes) to this question causes installation to continue. And the final message will be:

```
Installation of SWxxxxxxx xxxx.xxxx completed.
```

For the TriCore the directory **ctri** will be created.

Step 6

For tape install: remove the temporary installation directory with the following commands:

```
cd /tmp  
rm -rf instdir
```

Step 7

If you purchased a protected TASKING product, license the software product as explained in section 1.5, *Licensing TASKING Products*.

Step 8

Logout.

1.4.1 SETTING THE ENVIRONMENT

After you have installed the software, you have to set some environment variables to make invocation of the tools easier. A list of all environment variables used by the toolchain is present in the section *Environment Variables* in the chapter *Overview*.

Make sure that your path is set to include all of the executables you have just installed.

The environment variable TMPDIR can be used to specify a directory where programs can place temporary files.

1.5 LICENSING TASKING PRODUCTS

TASKING products are protected with license management software (FLEXlm). To use a TASKING product, you must install the licensing information provided by TASKING for the type of license purchased.

You can run TASKING products with a node-locked license or with a floating license. When you order a TASKING product determine which type of license you need (UNIX products only have a floating license).

Node-locked license (PC only)

This license type locks the software to one specific PC so you can use the product on that particular PC only.

Floating license

This license type manages the use of TASKING product licenses among users at one site. This license type does not lock the software to one specific PC or workstation but it requires a network. The software can then be used on any computer in the network. The license specifies the number of users who can use the software simultaneously. A system allocating floating licenses is called a **license server**. A license manager running on the license server keeps track of the number of users.



See the *Flexible License Manager (FLEXlm)* appendix for detailed information on FLEXlm.

1.5.1 OBTAINING LICENSE INFORMATION

Before you can install a software license you must have a "License Information Form" containing the license information for your software product. If you have not received such a form follow the steps below to obtain one. Otherwise, you can install the license.

Node-locked license (PC only)

1. If you need a node-locked license, you must determine the hostid of the computer where you will be using the product. See section 1.5.7, *How to Determine the Hostid*.

2. When you order a TASKING product, provide the hostid to your local TASKING sales representative. The License Information Form which contains your license key information will be sent to you with the software product.

Floating license

1. If you need a floating license, you must determine the hostid and hostname of the computer where you want to use the license manager. Also decide how many users will be using the product. See section 1.5.7, *How to Determine the Hostid* and section 1.5.8, *How to Determine the Hostname*.
2. When you order a TASKING product, provide the hostid, hostname and number of users to your local TASKING sales representative. The License Information Form which contains your license key information will be sent to you with the software product.

1.5.2 INSTALLING NODE-LOCKED LICENSES

Keep your "License Information Form" ready. If you do not have such a form read section 1.5.1, *Obtaining License Information*, before continuing.

Step 1

Install the TASKING software product following the installation procedure described in section 1.2, *Installation for Windows*.

Step 2

Create a file called "license.dat" in the c:\flexlm directory, using an ASCII editor and insert the license information contained in the "License Information Form" in this file. This file is called the "license file". If the directory c:\flexlm does not exist, create the directory.



If you wish to install the license file in a different directory, see section 1.5.6, *Modifying the License File Location*.



If you already have a license file, add the license information to the existing license file. If the license file already contains any SERVER lines, you must use another license file. See section 1.5.6, *Modifying the License File Location*, for additional information.

The software product and license file are now properly installed.



See the *Flexible License Manager (FLEXlm)* appendix for more information on FLEXlm.

1.5.3 INSTALLING FLOATING LICENSES

Keep your "License Information Form" ready. If you do not have such a form read section 1.5.1, *Obtaining License Information*, before continuing.

Step 1

Install the TASKING software product following the installation procedure described earlier in this chapter on the computer or workstation where you will use the software product.

As a result of this installation two additional files for FLEXlm will be present in the `flexlm` subdirectory of the toolchain:

Tasking	The Tasking daemon (vendor daemon).
license.dat	A template license file.

Step 2

If you already have installed FLEXlm v6.1 or higher for Windows or v2.4 or higher for UNIX (for example as part of another product) you can skip this step and continue with step 3. Otherwise, install SW000098, the Flexible License Manager (FLEXlm), on the license server where you want to use the license manager.

The installation of the license manager on Windows also sets up the license daemon to run automatically whenever a license server reboots. On UNIX you have to perform the steps as described in section 1.5.5, *Setting Up the License Deaemon to Run Automatically*.



It is not recommended to run a license manager on a Windows 95 or Windows 98 machine. Use Windows NT instead (or UNIX).

Step 3

If FLEXlm has already been installed as part of a non-TASKING product you have to make sure that the `bin` directory of the FLEXlm product contains a copy of the **Tasking** daemon (see step 1).

Step 4

Insert the license information contained in the "License Information Form" in the license file, which is being used by the license server. This file is usually called `license.dat`. The default location of the license file is in directory `c:\flexlm` for Windows and in `/usr/local/flexlm/licenses` for UNIX.



If you wish to install the license file in a different directory, see section 1.5.6, *Modifying the License File Location*.

If the license file does not exist, you have to create it using an ASCII editor. You can use the license file `license.dat` from the toolchain's `flexlm` subdirectory as a template.



If you already have a license file, add the license information to the existing license file. If the SERVER lines in the license file are the same as the SERVER lines in the License Information Form, you do not need to add this same information again. If the SERVER lines are not the same, you must use another license file. See section 1.5.6, *Modifying the License File Location*, for additional information.

Step 5

On each PC or workstation where you will use the TASKING software product the location of the license file must be known. If it differs from the default location (`c:\flexlm\license.dat` for Windows, `/usr/local/flexlm/licenses/license.dat` for UNIX), then you must set the environment variable **LM_LICENSE_FILE**. See section 1.5.6, *Modifying the License File Location*, for more information.

Step 6

Now all license information is entered, the license manager must be started (see section 1.5.4). Or, if it is already running you must notify the license manager that the license file has changed by entering the command (located in the `flexlm bin` directory):

lmreread

On Windows you can also use the graphical FLEXlm Tools (**lmttools**): Start **lmttools** (if you have used the defaults this can be done by selecting Start | Programs | TASKING FLEXlm | FLEXlm Tools), fill in the current license file location if this field is empty, click on the Reread button and then on OK. Another option is to reboot your PC.

The software product and license file are now properly installed.

Where to go from here?

The license manager (daemon) must always be up and running. Read section 1.5.4 on how to start the daemon and read section 1.5.5 for information how to set up the license daemon to run automatically.

If the license manager is running, you can now start using the TASKING product.



See the *Flexible License Manager (FLEXlm)* appendix for detailed information on FLEXlm.

1.5.4 STARTING THE LICENSE DAEMON

The license manager (daemon) must always be up and running. To start the daemon complete the following steps on each license server:

Windows

1. Start the license manager tool by (Start | Programs | TASKING FLEXlm | FLEXlm License Manager).
2. In the Control tab, click on the Start button.
3. Close the program by clicking on the OK button.

UNIX

1. Log in as the operating system administrator (usually root).
2. Change to the FLEXlm installation directory (default /usr/local/flexlm):

```
cd /usr/local/flexlm
```

3. For C shell users, start the license daemon by typing the following:

```
bin/lmgrd -2 -p -c licenses/license.dat >>& \  
/var/tmp/license.log &
```


Or, for Bourne shell users, start the license daemon by typing the following:

```
bin/lmgrd -2 -p -c licenses/license.dat >> \
/var/tmp/license.log 2>&1 &
```

In these two commands, the **-2** and **-p** options restrict the use of the **lmdown** and **lmremove** license administration tools to the license administrator. You omit these options if you want. Refer to the usage of **lmgrd** in the *Flexible License Manager (FLEXlm)* appendix for more information.

1.5.5 SETTING UP THE LICENSE DAEMON TO RUN AUTOMATICALLY

To set up the license daemon so that it runs automatically whenever a license server reboots, follow the instructions below that are appropriate for your platform. steps on each license server:

Windows

1. Start the license manager tool by (Start | Programs | TASKING FLEXlm | FLEXlm License Manager).
2. In the Setup tab, enable the Start Server at Power-Up check box.
3. Close the program by clicking on the OK button. If a question appears, answer Yes to save your settings.

UNIX



In performing any of the procedures below, keep in mind the following:

- Before you edit any system file, make a backup copy.

HP-UX

1. Log in as the operating system administrator (usually root).
2. In the directory `/etc/rc.config.d` create a file named `rc.lmgrd` with the following contents. Replace *FLEXLMDIR* by the FLEXlm installation directory (default `/usr/local/flexlm`):

```
#!/sbin/sh
FLEXLMDIR/bin/lmgrd -2 -p -c FLEXLMDIR/licenses/license.dat >> \
/var/tmp/license.log 2>&1 &
```

After the **-c** option, you have to specify the correct location of the license file.

SunOS4

1. Log in as the operating system administrator (usually root).
2. Append the following lines to the file `/etc/rc.local`. Replace *FLEXLMDIR* by the FLEXlm installation directory (default `/usr/local/flexlm`):

```
FLEXLMDIR/bin/lmgrd -2 -p -c FLEXLMDIR/licenses/license.dat >> \
/var/tmp/license.log 2>&1 &
```

SunOS5 (Solaris 2)

1. Log in as the operating system administrator (usually root).
2. In the directory `/etc/init.d` create a file named `rc.lmgrd` with the following contents. Replace *FLEXLMDIR* by the FLEXlm installation directory (default `/usr/local/flexlm`):

```
#!/bin/sh
FLEXLMDIR/bin/lmgrd -2 -p -c FLEXLMDIR/licenses/license.dat >> \
/var/tmp/license.log 2>&1 &
```

3. Make it executable:

```
chmod u+x rc.lmgrd
```

4. Create an 'S' link in the `/etc/rc3.d` directory to this file and create 'K' links in the other `/etc/rc?.d` directories:

```
ln /etc/init.d/rc.lmgrd /etc/rc3.d/Snumrc.lmgrd
ln /etc/init.d/rc.lmgrd /etc/rc?.d/Knumrc.lmgrd
```

num must be an appropriate sequence number. Refer to your operating system documentation for more information.

1.5.6 MODIFYING THE LICENSE FILE LOCATION

The default location for the license file on Windows is:

```
c:\flexlm\license.dat
```

On UNIX this is:

```
/usr/local/flexlm/licenses/license.dat
```

If you want to use another name or directory for the license file, each user must define the environment variable **LM_LICENSE_FILE**. Do this in `autoexec.bat` (Windows 95/98), from the Control Panel -> System | Environment (Windows NT) or in a UNIX login script.

If you have more than one product using the FLEXlm license manager you can specify multiple license files to the **LM_LICENSE_FILE** environment variable by separating each pathname (*lppath*) with a ';' (on UNIX also ':'):

Example Windows:

```
set LM_LICENSE_FILE=c:\flexlm\license.dat;c:\license.txt
```

Example UNIX:

```
setenv LM_LICENSE_FILE
/usr/local/flexlm/licenses/license.dat:/myprod/license.txt
```

If the license file is not available on these hosts, you must set **LM_LICENSE_FILE** to *port@host*; where *host* is the host name of the system which runs the FLEXlm license manager and *port* is the TCP/IP port number on which the license manager listens.

To obtain the port number, look in the license file at *host* for a line starting with "SERVER". The fourth field on this line specifies the TCP/IP port number on which the license server listens. For example:

```
setenv LM_LICENSE_FILE 7594@elliott
```



See the *Flexible License Manager (FLEXlm)* appendix for detailed information.

1.5.7 HOW TO DETERMINE THE HOSTID

The hostid depends on the platform of the machine. Please use one of the methods listed below to determine the hostid.

Platform	Tool to retrieve hostid	Example hostid
HP-UX	lanscan (use the station address without the leading '0x')	0000F0050185
SunOS/Solaris	hostid	170a3472
Windows	tkhostid (or use lmhostid)	0800200055327

Table 1-1: Determine the hostid



If you do not have the program **tkhostid** you can download it from our Web site at: <http://www.tasking.com/support/flexlm/tkhostid.zip> . It is also on every product CD that includes FLEXlm.

1.5.8 HOW TO DETERMINE THE HOSTNAME

To retrieve the hostname of a machine, use one of the following methods.

Platform	Method
HP-UX	hostname
SunOS/Solaris	hostname
Windows 95/98	Go to the Control Panel, open "Network", click on "Identification". Look for "Computer name".
Windows NT	Go to the Control Panel, open "Network". In the "Identification" tab look for "Computer Name".

Table 1-2: Determine the hostname

INSTALLATION

CHAPTER

2

OVERVIEW



2

CHAPTER

2.1 INTRODUCTION TO TRICORE C CROSS-COMPILER

This manual provides a functional description of the TASKING TriCore C Cross-Compiler. This manual uses **ctri** (the name of the binary) as a shorthand notation for "TASKING TriCore C Compiler".

TASKING offers a complete toolchain for the Siemens TriCore family of processors. 'TriCore' is used as a shorthand notation for the TriCore family of processors and their derivatives.

The TASKING TriCore C compiler accepts source programs written in ANSI C and translates these into TriCore assembly source code files. The compiler accepts language extensions to improve code performance and to allow the use of typical TriCore architectural provisions efficiently at the C level. The compiler is ANSI C compatible and consists of three major parts; the *preprocessor*, the C *frontend* and the associated TriCore *backend* or code generator. These are all integrated into a single program to avoid the need of intermediate files, thus speeding up the compilation process. It also simplifies the implementation of joint frontend-backend optimization strategies and preprocessor pragmas. This effectively makes the compiler a one pass compiler, with minimum file I/O overhead.

The compiler processes one C function at a time, until the entire source module has been read. The function is parsed, checked on semantic correctness and then transformed into an intermediate code tree that is stored in memory. Code optimizations are performed during the construction of the intermediate code, and are also applied when the complete function has been processed. The latter are often referred to as *global* optimizations.

ctri generates assembly source code using the TriCore assembly language specification, you must assemble this code with the TASKING TriCore Cross-Assembler. This manual uses **astri** as a shorthand notation for "TASKING TriCore Cross-Assembler".

You can link the generated object with other objects and libraries using the TASKING **lktri** TriCore linker. In this manual we use **lktri** as a shorthand notation for "TASKING **lktri** TriCore linker". You can locate the linked object to a complete application using the TASKING **lctri** TriCore locator. In this manual we use **lctri** as a shorthand notation for "TASKING **lctri** TriCore locator".

The program **cctri** is a control program. The control program facilitates the invocation of various components of the TriCore toolchain. **cctri** recognizes several filename extensions. C source files (.c) are passed to the compiler. Assembly sources (.asm and .src) are passed to the assembler. Relocatable object files (.obj) and libraries (.a) are recognized as linker input files. Files with extension .out and .dsc are treated as locator input files. The control program supports options to stop at any stage in the compilation process and has options to produce and retain intermediate files.

You can debug the software written in C with the TASKING CrossView Pro high-level language debugger. A list of supported platforms and emulators is available from TASKING.

2.2 PRODUCT DEFINITION

Name:

TASKING TriCore C Cross-Compiler

Ordering Code:

TK060002

Target Assembler:

TASKING TriCore Cross-Assembler
TK060000 (included in TK060002)

Target Debugger:

TASKING TriCore CrossView Pro debugger (TK060043)

Target Processors:

All TriCore derivatives. Special function registers can be accessed by means of a user-definable register file.

2.3 GENERAL IMPLEMENTATION

This section describes the different phases of the compiler and the target independent optimizations.

2.3.1 COMPILER PHASES

During the compilation of a C program, a number of phases can be identified. These phases are divided into two groups, referred to as *frontend* and *backend*.

frontend:

The preprocessor phase:

File inclusion and macro substitution are done by the preprocessor before parsing of the C program starts. The syntax of the macro preprocessor is independent of the C syntax, but also described in the ANSI X3.159-1989 standard.

The scanner phase:

The scanner converts the preprocessor output to a stream of tokens.

The parser phase:

The tokens are fed to a parser for the C grammar. The parser performs a syntactic and semantic analysis of the program, and generates an intermediate representation of the program.

The frontend optimization phase:

Target processor independent optimization is performed by transforming the intermediate code. The next section discusses the frontend optimizations.

backend:

The backend optimization phase:

Performs target processor specific optimizations. Very often this means another transformation of the intermediate code and actions like register allocation techniques for variables, expression evaluation and the best usage of the addressing modes. The chapter *Language Implementation* discusses this item in more detail.

The code generator phase:

This phase converts the intermediate code to an internal instruction code, representing the TriCore assembly instructions.

The peephole optimizer / pipeline scheduler phase:

This phase uses pattern matching techniques to perform peephole optimizations on the internal code. The pipeline scheduler reorders and combines instructions to minimize the number of instructions. Finally the peephole optimizer translates the internal instruction code into assembly code for **astri**. The generated assembly does not contain any macros. The assembler is also equipped with an optimizer.

All phases (of both frontend and backend) of the compiler are combined into one program. The compiler does not use intermediate files for communication between the different phases of compilation. The backend part is not called for each C statement, but starts after a complete C function has been processed by the frontend (in memory), thus allowing more optimization. The compiler only requires one pass over the input file, resulting in relatively fast compilation.

2.3.2 FRONTEND OPTIMIZATIONS

The command line option **-O** controls the amount of optimization applied on the C source. Within a source file, the pragma `#pragma optimize` sets the optimization level of the compiler. Using the pragma, certain optimizations can be switched on or off for a particular part of the program. Several optimizations cannot be controlled individually. e.g., constant folding will always be done.

The compiler performs the following optimizations on the intermediate code. They are independent of the target processor and the code generation strategy:

Constant folding

Expressions only involving constants are replaced by their result. E.g. $1+5*3$ is replaced by the value 16.

Expression rearrangement

Expressions are rearranged to allow more constant folding. E.g. $5 + (x-3)$ is transformed into $x + (5-3)$, which can be folded and will result in $x+2$.

Expression simplification

As an example, multiplication by 0 or 1 and additions or subtractions of 0 are removed. Such useless expressions can be introduced by macros, or by the compiler itself (e.g., array subscription).

Logical expression optimization

Expressions involving '&&', '||' and '!' are interpreted and translated into a series of conditional jumps.

Loop rotation

With `for` and `while` loops, the expression is evaluated once at the 'top' and then at the 'bottom' of the loop. This optimization speeds up execution, and it could also save code in some cases.

Control flow optimization

By reversing jump conditions and moving code, the number of jump instructions is minimized. This reduces both the code size and the execution time.

Jump chaining

A conditional or unconditional jump to a label which is immediately followed by an unconditional jump may be replaced by a jump to the destination label of the second jump. This optimization does not save code, but speeds up execution.

Remove useless jumps

An unconditional jump to a label directly following the jump is removed. A conditional jump to such a label is replaced by an evaluation of the jump condition. The evaluation is necessary because it may have side effects.

Conditional jump reversal

A conditional jump over an unconditional jump is transformed into one conditional jump with the jump condition reversed. This reduces both the code size and the execution time.

Constant/copy propagation

A reference to a variable with known contents is replaced by those contents.

Common subexpression elimination

The compiler has the ability to detect repeated uses of the same (sub-) expression. Such a "common" expression may be temporarily saved to avoid recomputation. This method is called *common subexpression elimination*, abbreviated CSE.

Invariant code motion

Invariant code can be moved out of a loop.

Subscript strength reduction

Expressions involving a loop index variable can be reduced in strength.

Dead code elimination

Unreachable code can be removed from the intermediate code without affecting the program. However, the compiler generates a warning message, because the unreachable code may be the result of a coding error.

Dead assignment elimination

Removal of assignments to objects that are not used afterwards.

Dead storage elimination

Remove unused storage.

Loop unrolling

Eliminates short loops by replacing them with a number of copies.

In-line functions

In-line functions are supported. The overhead caused by a function call is removed.

Sharing of string literals and floating point constants

String literals and floating point constants are put in ROM memory. The compiler overlays identical strings (within the same module) and let them share the same space, thus saving ROM space. Likewise identical floating point constants are overlaid and allocated only once.

2.3.3 BACKEND OPTIMIZATIONS

The following optimizations are target dependent and are therefore performed by the backend.

Peephole optimizations

The generated assembly code is improved by replacing instruction sequences by equivalent but faster and/or shorter sequences, or by deleting unnecessary instructions.

Leaf function handling

Leaf functions (function not calling other functions), are handled specially with respect to stack frame building.

Tail recursion elimination

Replace a recursion statement to branch to the beginning of the statement.

2.4 COMPILER STRUCTURE

If you want to build a TriCore application you need to invoke the following programs directly, or via the control program:

- The C compiler (**ctri**), which generates an assembly source file from the file with suffix `.c`. The suffix of the compiler output file is `.src`. However, you can direct the output to `stdout` with the **-n** option, or to another file with the **-o** option. C source lines can be intermixed with the generated assembly statements with the **-s** option. High level language debugging information can be generated with the **-g** option. You are advised to switch off all debugging with the **-gn** option when inspecting the generated assembly source code, because it contains a lot of 'unreadable' high level language debug directives. The C compilers make only one pass on every file. This pass checks the syntax, generates the code and performs code optimization.
- The corresponding cross-assembler (**astri**), which processes the generated assembly source file into a relocatable object file with suffix `.obj`.
- The **lktri** linker, which links the generated relocatable object files and C-libraries. The result is a relocatable object file with suffix `.out`. A linker map file with suffix `.lnl` is available after this stage.
- The **lctri** locator, which locates the generated relocatable object files. The result is an absolute loadable file with suffix `.abs`. A full application map file with suffix `.map` is available after this stage.

You can directly load the output file of the locator with extension `.abs` into the CrossView Pro debugger.

The next figure explains the relationship between the different parts of the TASKING TriCore toolchain:

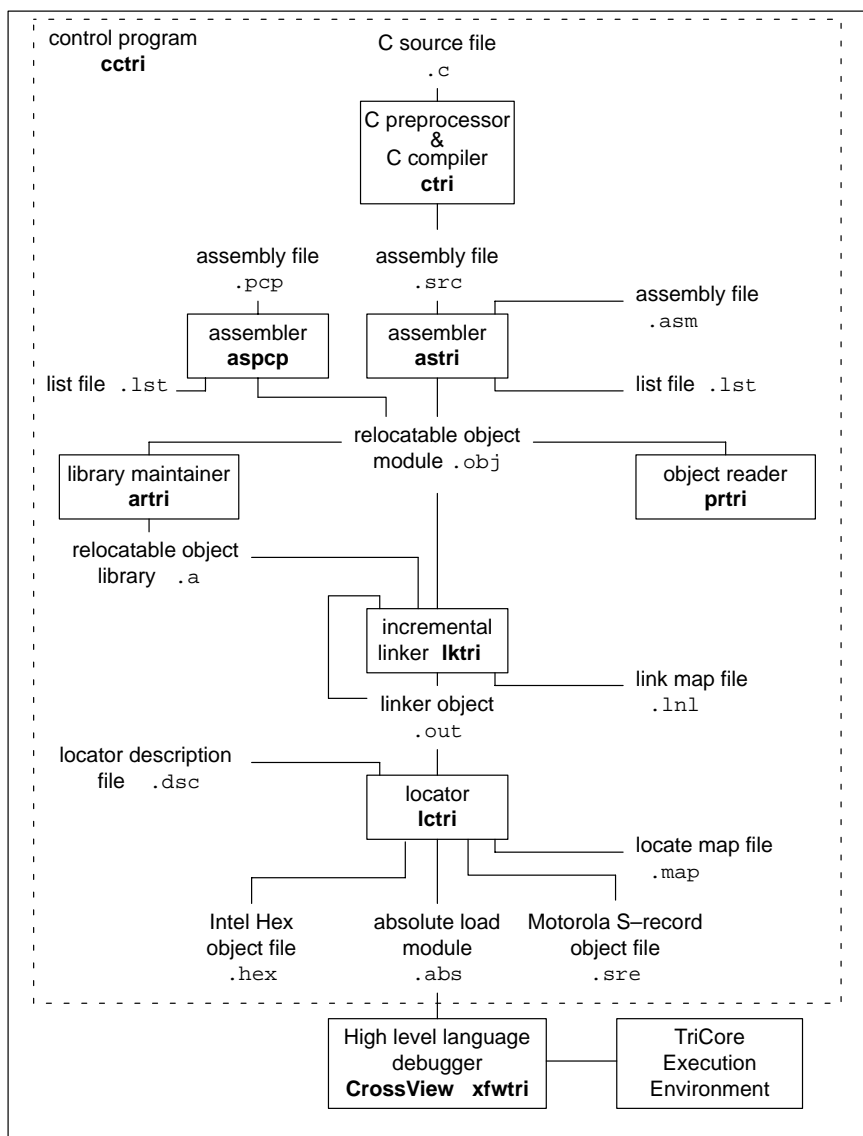


Figure 2-1: TriCore development flow

The program **cctri** is a so-called control program, which facilitates the invocation of various components of the TriCore toolchain. C source programs are compiled by the compiler, assembly source files are passed to the assembler. A C preprocessor program is available as an integrated part of the C compiler. The control program recognizes the file extensions **.a** and **.obj** as input files for the linker. The control program passes files with extensions **.out** and **.dsc** to the locator. All other files are considered to be object files and are passed to the linker. The control program has options to suppress the locating stage (**-cl**), the linker stage (**-c**) or the assembler stage (**-cs**).

Optionally the locator, **lctri** produces output files in Motorola S-record format or Intel Hex format. The default output format is IEEE-695.

Normally, the control program removes intermediate compilation results, as soon as the next phase completes successfully. If you want to retain all intermediate files, the option **-tmp** prevents removal of these files.

For a description of all utilities available and the possible output formats of the locator, see the TriCore Cross-Assembler User's Guide.

The name of the TriCore CrossView Pro Debugger is **xfwtri**. For more information check the TriCore CrossView Pro Debugger User's Guide.

2.5 ENVIRONMENT VARIABLES

This section contains an overview of the environment variables used by the TriCore toolchain.

Environment Variable	Description
ASTRIINC	Specifies an alternative path for include files for the assembler.
CTRIINC	Specifies an alternative path for #include files for the C compiler ctri .
CTRLIB	Specifies a path to search for library files used by the linker lktri .
CCTRIBIN	When this variable is set, the control program, cctri , prepends the directory specified by this variable to the names of the tools invoked.
CCTRIOPT	Specifies extra options and/or arguments to each invocation of cctri . The control program processes the arguments from this variable before the command line arguments.
LM_LICENSE_FILE	Identifies the location of the license data file. Only needed for hosts that need the FLEXlm license manager.
PATH	Specifies the search path for your executables.
TMPDIR	Specifies an alternative directory where programs can create temporary files. Used by ctri , cctri , astri , lktri , lctri , artri .

Table 2-1: Environment variables

2.6 SAMPLE SESSION

The subdirectory `dhry` in the `examples` subdirectory contains a demo program for the TriCore toolchain.

In order to debug your programs, you will have to compile, assemble, link and locate them for debugging using the TASKING TriCore tools. You can do this with one call to the control program or you can use EDE, the Embedded Development Environment (which uses a project file and a makefile) or you can call the makefile from the command line.

2.6.1 USING EDE

EDE stands for "Embedded Development Environment" and is the MS-Windows oriented Integrated Development Environment you can use with your TASKING toolchain to design and develop your application.

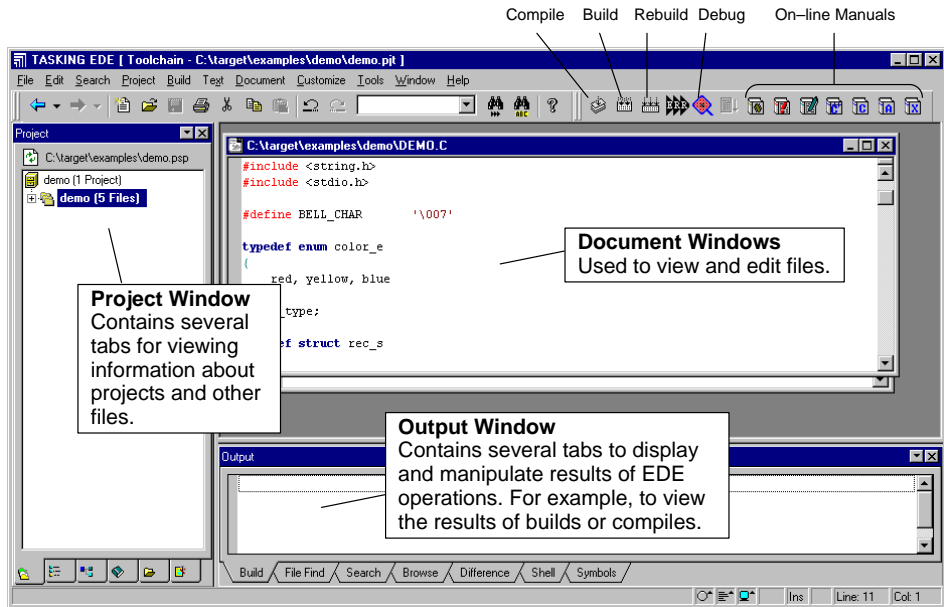
To use EDE on the `dhry` demo program in the subdirectory `dhry` in the `examples` subdirectory of the TriCore product tree follow the steps below. This procedure is outlined as a guide for you to build your own executables for debugging.

How to Start EDE

You can launch EDE by double-clicking on the EDE shortcut on your desktop.



The EDE screen provides you with a menu bar, a toolbar (command buttons) and one or more windows (for example, for source files), a status bar and numerous dialog boxes.



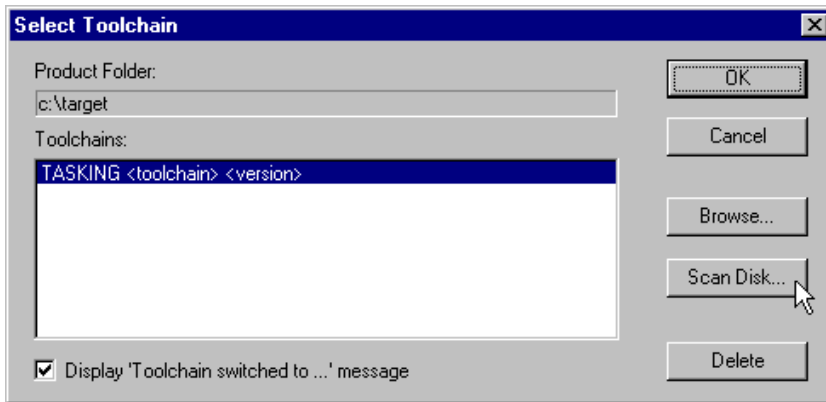
How to Select a Toolchain

EDE supports all the TASKING toolchains. When you first start EDE, the correct toolchain of the product you purchased is selected and displayed in the title of the EDE desktop window.

If you have more than one TASKING product installed and you want to change toolchains, do the following::

1. From the Project menu, select Select Toolchain...

The Select Toolchain dialog appears.



2. Select the toolchain you want. You can do this by clicking on a toolchain in the Toolchains list box and click OK.

If no toolchains are present, use the Browse... or Scan Disk... button to search for a toolchain directory. Use the Browse... button if you know the installation directory of another TASKING product. Use the Scan Disk... button to search for all TASKING products present on a specific drive. Then return to step 2.

How to Open an Existing Project

Follow these steps to open an existing project:

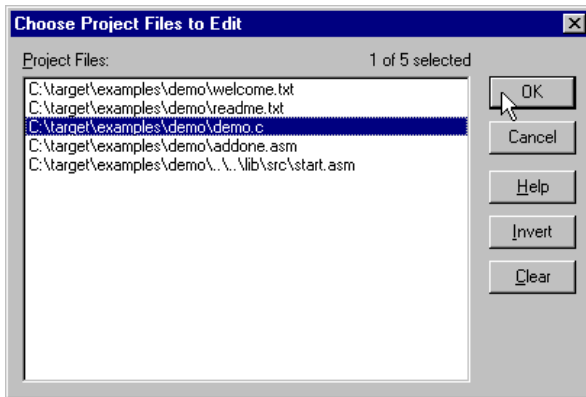
1. From the Project menu, select Set Current ->.
2. Select the project file to open. For the dhry demo program select the file dhry_1.pjt in the subdirectory dhry in the examples subdirectory of the TriCore product tree. If you have used the defaults, the file dhry_1.pjt is in the directory c:\ctri\examples\dhry.

How to Load/Open Files

The next two steps are not needed for the demo program because the files dhry_1.c and dhry_2.c are already open. To load the file you want to look at:

1. From the Project menu, select Load Files...

The Choose Project Files to Edit dialog appears.



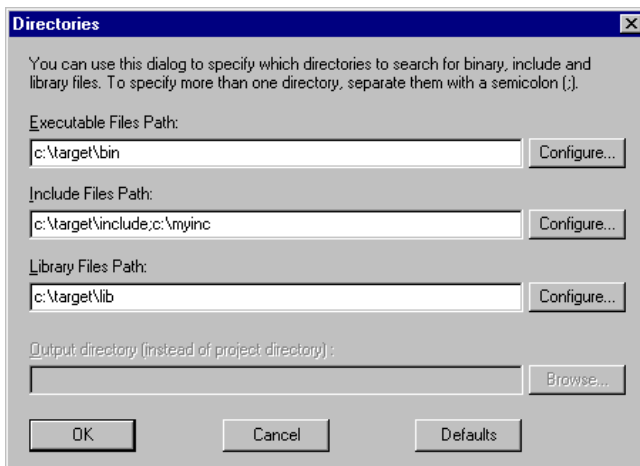
2. Choose the file(s) you want to open by clicking on it. You can select multiple files by pressing the <Ctrl> or <Shift> key while you click on a file. With the <Ctrl> key you can make single selections and with the <Shift> key you can select everything from the first selected file to the file you click on. Then click OK.

This launches the file(s) so you can edit it (them).

Check the directory paths

1. From the Project menu, select Directories....

The Directories dialog appears.



2. Check the directory paths for programs, include files and libraries. You can add your own directories here, separated by semicolons.
3. Click OK.

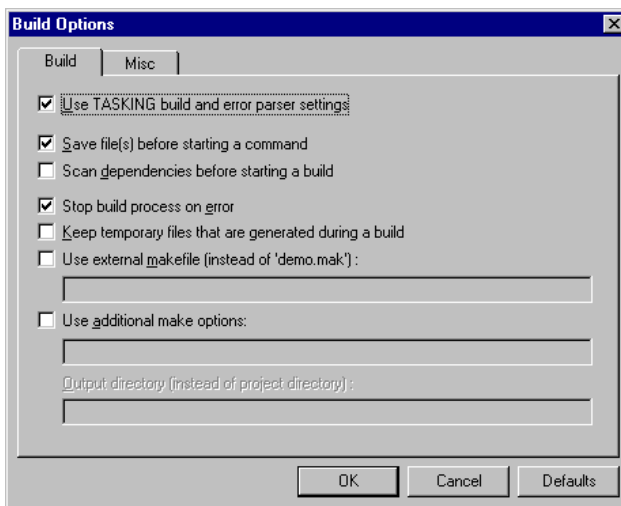
How to Build the Demo Application

The next step is to compile the file(s) together with its dependent files so you can debug the application.

Steps 1 and 2 are optional. Follow these steps if you want to specify additional build options such as to stop the build process on errors and to keep temporary files that are generated during a build.

1. From the Build menu, select Options...

The Build Options dialog appears.



2. Make your changes and press the OK button.
3. From the Build menu, select Scan All Dependencies.
4. Click on the Execute 'Make' command button. The following button is the execute Make button which is located in the ribbon bar.



If there are any unsaved files, EDE will ask you in a separate dialog if you want to save them before starting the build.

How to View the Results of a Build

Once the files have been processed you can inspect the generated messages.

You can see which commands (and corresponding output captured) which have been executed by the build process in the Build tab:

```
TASKING program builder vx.y rz      Build nnn SN 00000000
Compiling "dhry_2.c"
Assembling "dhry_2.src"
Compiling "dhry_1.c"
Assembling "dhry_1.src"
Linking and locating to dhry_1.out
Converting dhry_1.out to dhry_1.abs in IEEE-695 format
.
.
.
```

How to Start the CrossView Pro Debugger

Once the files have been compiled, assembled, linked, located and formatted they can be executed by CrossView Pro.

To execute CrossView Pro:

1. Click on the `Debug` application button. The following button is the `Debug` application button which is located in the toolbar.



CrossView Pro is launched. CrossView Pro will automatically download the compiled file for debugging.

How to Load an Application

You must tell CrossView Pro which program you want to debug:

1. From the `File` menu, select `Load Symbolic Debug Info...`

The `Load Symbolic Debug Info` dialog box appears.

2. Click `Load`.

How to View and Execute an Application

To view your source while debugging, the Source Window must be open.
To open this window:

1. From the View menu, select `Source->Source lines`.

The source window opens.

Before starting execution you have to reset the target system to its initial state. The program counter, stack pointer and any other registers must be set to their initial value. The easiest way to do this is:

2. From the Run menu, select `Reset Target System`.

To run your application step-by-step:

3. From the Run menu, select `Animate`.

The program `dhry_1.abs` is now stepping through the high level language statements. Using the Accelerator bar or the menu bar you can set breakpoints, monitor data, display registers, simulate I/O and much more. See the *CrossView Pro Debugger User's Guide* for more information.

How to Start a New Project

When you first use EDE you need to setup a project space and add a new project:

1. From the File menu, select `New Project Space...`

The `Create a New Project Space` dialog appears.

2. Give your project space a name and then click OK.

The `Project Properties` dialog box appears.

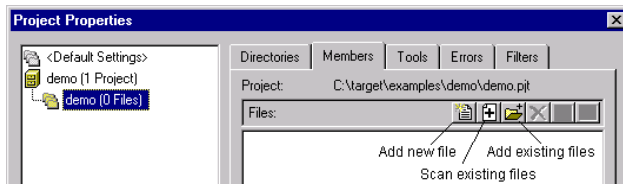
3. Click on the `Add new project to project space` button.

The `Add New Project to Project Space` dialog appears.

4. Give your project a name and then click OK.

The `Project Properties` dialog box then appears for you to identify the files to be added.

5. Add all the files you want to be part of your project. Then press the OK button. To add files, use one of the 3 methods described below.



- If you do not have any source files yet, click on the **Add new file** to project button in the **Project Properties** dialog. Enter a new filename and click OK.
- To add existing files to a project by specifying a file pattern click on the **Scan existing files into project** button in the **Project Properties** dialog. Select the directory that contains the files you want to add to your project. Enter one or more file patterns separated by semicolons. The button next to the **Pattern** field contains some predefined patterns. Next click OK.
- To add existing files to a project by selecting individual files click on the **Add existing files to project** button in the **Project Properties** dialog. Select the directory that contains the files you want to add to your project. Add the applicable files by double-clicking on them or by selecting them and pressing the **Open** button.

The new project is now open.

6. From the **Project** menu, select **Load Files...** to open the files you want on your EDE desktop.

EDE automatically creates a makefile for the project. EDE updates the makefile every time you modify your project.

2.6.2 USING THE CONTROL PROGRAM

A detailed description of the process using the sample program `dhry` is described below. This procedure is outlined as a guide for you to build your own executables for debugging.

1. Make the subdirectory `dhry` of the `examples` directory the current working directory.
2. Be sure that the directory of the binaries is present in the `PATH` environment variable.

3. Compile, assemble, link and locate the modules using one call to the control program **cctri**:

```
cctri -g -M -o dhry_1.abs dhry_1.c dhry_2.c
```

The **-g** option specifies to generate symbolic debugging information. This option must always be specified when debugging with CrossView Pro.

The **-M** option specifies to generate map files.

The **-o** option specifies the name of the output file.

The command in step 3 generates the object files `dhry_1.obj` and `dhry_2.obj`, the linker map files `dhry_1.lnl` and `dhry_2.lnl`, the locator map files `dhry_1.map` and `dhry_2.map` and the absolute output file `dhry_1.abs`. The file `dhry_1.abs` is in the IEEE Std. 695 format, and can directly be used by CrossView. No separate formatter is needed.

Now you have created all the files necessary for debugging with CrossView Pro with one call to the control program.

If you want to see how the control program calls the compiler, assembler, linker and locator, you can use the **-v0** option or **-v** option. The **-v0** option only displays the invocations without executing them. The **-v** option also executes them.

```
cctri -g -M -o dhry_1.abs dhry_1.c dhry_2.c -v0
```

The control program shows the following command invocations without executing them (UNIX output):

```
dhry_1.c:
+ ctri -e -g -o /tmp/cc2154b.src dhry_1.c
+ astri /tmp/cc2154b.src -e -ghl -o dhry_1.obj
dhry_2.c:
+ ctri -e -g -o /tmp/cc2154c.src dhry_2.c
+ astri /tmp/cc2154c.src -e -ghl -o dhry_2.obj
+ lktri -e -M dhry_1.obj dhry_2.obj -lc -lfpn -Ltc1
-o/tmp/cc2154d.out
+ lctri -e -M -dtri.dsc /tmp/cc2154d.out -odhry_1.abs
```

The **-e** option removes output files after errors occur. The **-gsl** option of the assembler specifies to pass HLL debug information and to generate local symbols debug information. The **-O** option of the linker specifies the basename of the map file. The **-lc** options of the linker specifies to link the appropriate C library. The **-d** option of the locator specifies the name of the locator description file.

As you can see, the tools use temporary files for intermediate results. If you want to keep the intermediate files you can use the **-tmp** option. The following command makes this clear.

```
cctri -g -M -o dhry_1.abs dhry_1.c dhry_2.c -v0 -tmp
```

This command produces the following output:

```
dhry_1.c:
+ ctri -e -g -o dhry_1.src dhry_1.c
+ astri dhry_1.src -e -ghl -o dhry_1.obj
dhry_2.c:
+ ctri -e -g -o dhry_2.src dhry_2.c
+ astri dhry_2.src -e -ghl -o dhry_2.obj
+ lktri -e -M dhry_1.obj dhry_2.obj -lc -lfpn -ltcl
-odhry_1.out
+ lctri -e -M -dtri.dsc dhry_1.out -odhry_1.abs
```

As you can see, if you use the **-tmp** option, the assembly source files and linker output file will be created in your current directory also.

Of course, you will get the same result if you invoke the tools separately using the same calling scheme as the control program.

As you can see, the control program automatically calls each tool with the correct options and controls. The control program is described in detail in Chapter *Compiler Use*.

2.6.3 USING THE MAKEFILE

The subdirectories in the `examples` directory each contain a `makefile` which can be processed by **mktri**. Also each subdirectory contains a `readme.txt` file with a description of how to build the example.

To build the `dhry` demo example follow the steps below. This procedure is outlined as a guide for you to build your own executables for debugging.

1. Make the subdirectory `dhry` of the `examples` directory the current working directory.

This directory contains a makefile for building the `dhry` demo example. It uses the default **mktri** rules.

2. Be sure that the directory of the binaries is present in the `PATH` environment variable.
3. Compile, assemble, link and locate the modules using one call to the program builder **mktri**:

```
mktri
```

This command will build the example using the file `makefile`.

To see which commands are invoked by **mktri** without actually executing them, type:

```
mktri -n
```

This command produces the following output:

```
TASKING TriCore program builder    vx.yrz Build nnn
Copyright 1996-year Altium BV      Serial# 00000000
cctri -c -o dhry_1.obj -g -w91 -w303 dhry_1.c
cctri -c -o dhry_2.obj -g -w91 -w303 dhry_2.c
cctri -o dhry_1.abs dhry_1.obj dhry_2.obj
```

The **-g** option in the makefile is used to instruct the C compiler to generate symbolic debug information. This information makes debugging an application written in C much easier to debug.

The **-w** option in the makefile is used to suppress a warning.

The **-o** option specifies the name of the output file.

To remove all generated files type:

```
mktri clean
```

CHAPTER

3

LANGUAGE IMPLEMENTATION



3

CHAPTER

3.1 INTRODUCTION

The TASKING C cross-compiler (**ctri**) offers a new approach to high-level language programming for the TriCore family. It conforms to the ANSI standard, but allows you to control the special functions of the TriCore in C.

This chapter describes the C language implementation in relation to the TriCore architecture.

The extensions to the C language in **ctri** are:

additional data types

In addition to the standard data type, **ctri** supports three additional basic types to perform fixed point arithmetic (`_fract`, `_sfract` and `_accum`). Two additional basic types were added to the C compiler to support the packed arithmetic instructions (`_packb` and `_packhw`). The integral type `_bit` is added to support the bit instructions.

_at

You can specify a variable to be at an absolute address.

_atbit

You can specify a variable to be at a bit offset within a `_bitword` or bit-addressable `_sfr` variable.

bit fields

You can use the type qualifiers `_sfrbit16` and `_sfrbit32` to control the access of SFR bit fields.

storage types

Apart from a memory category (extern, static, ...) you can specify a storage type in each declaration (`_near`, `_far`, `_a0`, `_a1`, `_a8`, `_a9`).

interrupt functions

You can specify interrupt functions directly through interrupt vectors in the C language (`_interrupt` and `_interrupt_fast` keywords).

intrinsic functions

A number of pre-declared functions can be used to generate inline assembly code at the location of the intrinsic (built-in) function call. This avoids the overhead which is normally used to do parameter passing and context saving before executing the called function.

circular buffers

ctri supports circular buffers through the data type `_circ`.

3.2 ACCESSING MEMORY

In practice the majority of the C code of a complete application is standard C (without using any language extension). You can compile this part of the application without any modification, using the storage types which fits best to the requirements of the system (code density, amount of external RAM etc.).

Only a small part of the application uses language extensions. These parts often have some of the following properties. They

- access I/O, using the special function registers
- need high execution speed
- need high code density
- access non-default memory
- are used to service interrupts

3.2.1 STORAGE TYPES

Static storage specifiers can be used to allocate **static** objects in a particular memory area of the addressing space of the processor. All objects taking **static** storage may be declared with an explicit storage specifier. By default static variables will be allocated in `_near` or `_far` memory according to some heuristic rules (see **-N** option).

ctri recognizes the following storage type specifiers:

Storage Type	Description
<code>_near</code>	The data object must be directly addressable using the absolute addressing mode. The first 16K of each 256M block are directly addressable.
<code>_far</code>	The data must not be allocated in a direct addressable memory region.
<code>_a0</code>	The data is allocated in a section that is addressable with a sign-extended 16-bit offset from A0.
<code>_a1</code>	The data is allocated in a section that is addressable with a sign-extended 16-bit offset from A1.
<code>_a8</code>	The data is allocated in a section that is addressable with a sign-extended 16-bit offset from A8.
<code>_a9</code>	The data is allocated in a section that is addressable with a sign-extended 16-bit offset from A9.

Table 3-1: Storage type specifiers

Examples:

```
int _near Var_in_near; /* fast accessible integer
                       in directly addressable
                       memory */
int _near * _far Ptr_in_far_to_near; /* allocate pointer in
                                     _far memory, compiler will not
                                     use absolute addressing mode */
char _a0 string[] = "TriCore";      /* string in A0 memory */
```

Using the `_near` addressing qualifier, allows the compiler to generate faster access code for frequently used variables. Pointers are always 32-bit.

Functions are by default allocated in ROM Memory; the storage specifier may be omitted in that case. Also, function return values cannot be assigned to a storage area.

In addition to static storage specifiers, a static object can be assigned to a fixed memory address using the `_at()` keyword:

```
int myvar _at(0x100);
```

This is useful to interface to other programs using fixed memory schemes, or to access special function registers.

Examples using storage specifiers:

Some examples of using storage specifiers:

```
int _near *p; // pointer to int in _near memory
              // (pointer has 32-bit size)
int _far *g;  // pointer to int in _far memory
              // (pointer has 32-bit size)

g = p;        /* the compiler issues a warning */
```

If a library function declares:

```
extern int _near foo; //extern int in _near memory
```

and a data object is declared as:

```
int _far foo; //int in _far memory
```

the linker will flag this as an error. The usage of the variables is always without a storage specifier:

```
char _near example; // define a char in _near memory */
example = 2;        /* assign example */
```

The generated assembly would be:

```
movl6 d15,2
st.b example,d15
```

All allocations with the same storage specifiers are collected in units called 'sections'. The section with the `_near` attribute will be located within the first 16K of each 256M block. It is always possible to control the location of sections manually.

3.2.2 THE `_at()` ATTRIBUTE

In C for the TriCore it is possible to place certain variables at absolute addresses. Instead of writing a piece of assembly code, a variable can be placed on an absolute address using the `_at()` attribute.

Example:

```
unsigned char Display[80*24] _at( 0x2000 );
```

The example above creates a variable with the name `Display` at address `0x2000`. In the generated assembly code an absolute section will appear. On this position space is reserved for the variable `Display`.

A number of restrictions are in effect when placing variables on an absolute address:

- Only global variables can be placed on absolute addresses. Parameters of functions, or automatics within functions cannot be placed on an absolute address.
- When declared `extern`, the variable is not allocated by the compiler. When the same variable is allocated within another module but on a different address, the compiler, assembler or linker will not notice, because an assembler external object cannot specify an absolute address.
- When the variable is declared `static`, no public symbol will be generated (normal C behavior).
- Functions cannot be declared absolute.
- Absolute variables cannot overlap each other, declaring two absolute variables on the same address will cause an error generated by the assembler or by the linker. The compiler does not check this.
- Declaring the same absolute variable within two modules will also produce conflicts during link time (except when one of the modules declares the variable 'extern').

3.2.3 THE `_ATBIT()` ATTRIBUTE

In C for the TriCore it is possible to define bit variables within an int variable. This can be done with the `_atbit()` attribute. The syntax is:

```
_atbit( name, offset )
```

name is the name of a int variable and *offset* (range 0–31) is the bit–offset within the variable. The int variable must be defined with the `_at()` attribute to make the `_atbit` attribute valid.

Examples:

```
int          bw    _at(0x...);
_bit         myb   _atbit( bw, 3 );
```

Variable `bw` must have the `_at` attribute to make `_atbit` valid.

3.3 DATA TYPES

All ANSI C data types are supported. Three types of pointers are recognized. Object size and ranges:

Data Type	Size (in bytes)	Range
_bit	1	[0,1]
signed char	1	−128 to +127
unsigned char	1	0 to 255U
signed short	2	−32768 to +32767
unsigned short	2	0 to 65535U
signed int	4	−2147483648 to +2147483647
unsigned int	4	0 to 4294967295UL
signed long	4	−2147483648 to +2147483647
unsigned long	4	0 to 4294967295UL
float	4	+/- 1,1755E−38 to +/- 3,402E+38
double	8	+/- 2,225E−308 to +/- 1,798E+308
enum	4	0 to 4294967295
pointer	4	0 to 4294967295
_sfract	2	[−1,+1>
_fract	4	[−1,+1>
_accum	8	[−131072,+131071>

Table 3-2: Data types

- _bit, char, short, int and long are all integral types, supporting all implicit (automatic) conversions.
- the TriCore convention is used, storing variables with the most significant part at the higher memory address (Little Endian).
- float is implemented in little endian IEEE-754 32-bit single precision format.
- double is implemented in little endian IEEE-754 64-bit double precision format.

3.3.1 SIGNED CHARACTERS

The character type is treated as `signed char` by default. You can overrule this default with the `-u` command line option, which sets the default to unsigned char.

Examples:

The following declarations are identical when `-u` is not used.

```
char c;  
signed char c;
```

The following declarations are identical when `-u` is used.

```
char c;  
unsigned char c;
```

3.3.2 ANSI C TYPE CONVERSIONS

Integral promotions

According to the ANSI C X3.159-1989 standard, a character, a short integer, an integer bit field (either signed or unsigned), or an object of enumeration type, may be used in an expression wherever an integer may be used. If a `signed int` can represent all the values of the original type, then the value is converted to `signed int`; otherwise the value will be converted to `unsigned int`. This process is called *integral promotions* (as defined in the ANSI C standard).

In case of the Tricore this implies that among other things, all character and short integers, whether they are unsigned or signed, will be promoted to signed integer, when integral promotions is applied.

Integral promotions is also performed on function pointers and function parameters of integral types using the old-style declaration. To avoid problems with implicit type conversions, you are advised to use function prototypes.

Usual arithmetic conversions

Many operators cause conversions and yield result types in a similar way. The effect is to bring operands into a common type, which is also the type of the result. This pattern is called the *usual arithmetic conversions* (as defined in the ANSI C standard).

For the TriCore this means the following:

1. if necessary use long double, double or float (in this order).
2. (un)signed char and (un)signed short are mapped onto signed int.
3. if necessary use unsigned long, long or unsigned int (in this order)

The following pseudo code gives the conversion algorithm in more detail:

IF either operand is long double

 THEN the other operand is converted to long double

ELSE IF either operand is double

 THEN the other operand is converted to double

ELSE IF either operand is float

 THEN the other operand is converted to float

ELSE

 Integral promotions are performed on both operands;

 IF either operand is unsigned long

 THEN the other operand is converted to unsigned long.

 ELSE IF one operand is long and the other is unsigned int

 THEN

 IF a long can represent all values of an unsigned int

 THEN the unsigned int operand is converted to long

 ELSE both operands are converted to unsigned long.

 ELSE IF one operand is long

 THEN the other operand is converted to long.

 ELSE IF either operand is unsigned int.

 THEN the other operand is converted to unsigned int.

 ELSE both operands have type int.



Sometimes surprising results may occur, for example when an unsigned char is promoted to int. You can always use explicit casting to obtain the type required. The following example makes this clear:

```
static unsigned char a=0xFF, b, c;
void f()
{
    b=~a;
    if ( b == ~a )
    {
        /* This code is never reached,because
        * 0x00000000 is compared to 0xFFFFFFFF0.
        * The compiler converts character 'a' to an
        * int before applying the ~ operator,
        * so '~a' yields 0xFFFFFFFF0 in the
        * assignment '~a' is cast into an unsigned
        * char again, so the last byte is assigned
        * to b => b = 0x00 in the comparison '~a' is
        * again 0xFFFFFFFF0, b however is promoted to
        * an integer with value 0x00000000.
        */
        ...
    }
    c=a+1;
    while( c != a+1 )
    {
        /* This loop never stops because
        * 0x00000000 is compared to 0x00000100.
        * The compiler evaluates 'a+1' as an integer
        * expression, so 'a+1' is 0x000000FF +
        * 0x00000001, is 0x00000100.
        * In the assignment the last byte of the
        * result is assigned to c, so c = 0x00.
        * In the comparison both sides are promoted
        * to integer, the left side to 0x00000000
        * and the right side to 0x00000100 again.
        */
        ...
    }
}
```


To overcome this 'unwanted' behavior use an explicit cast:

```
static unsigned char a=0xFF, b, c;
void f()
{
    b=~a;
    if ( b == (unsigned char)~a )
    {
        /* This code is always reached */
        ...
    }
    c=a+1;
    while( c != (unsigned char)(a+1) )
    {
        /* This code is never reached */
        ...
    }
}
```

Keep in mind that the arithmetic conversions apply to multiplications also:

```
static int    h, i, j;
static long   k, l, m;

/* In C the following rules apply:
 *    int  * int    result: int
 *    long * long   result: long
 *
 * and NOT  int * int    result: long
 */
void f()
{
    h = i * j;           /* int  * int  = int          */
    k = l * m;           /* long * long = long        */

    l = i * j;           /* int  * int  = int,
                        * afterwards promoted (sign
                        * or zero extended) to long
                        */
    l = (long) i * j;     /* long * long = long        */
    l = (long)(i * j);    /* int  * int  = int,
                        * afterwards casted to long
                        */
}
```

3.4 FRACTIONAL DATA TYPES

3.4.1 ADDITIONAL BASIC TYPES

The TriCore C compiler supports three additional basic types to perform fixed point arithmetic. These types are:

_sfract

16 bits: 1 sign bit + 15 mantissa bits

_fract

32 bits: 1 sign bit + 31 mantissa bits

_accum

64 bits: 1 sign bit + 17 integral bits + 46 mantissa bits

Most basic operations on the first two types are directly supported by the TriCore instruction set. For example, the following assignment:

```
_sfract a, b, c;
...
a = b + c;
```

will result in the following assembly code:

```
ld.q    d15,b
ld.q    d7,c
add16   d15,d7
st.q    a,d15
```

3.4.2 TYPE CONVERSIONS

A conversion from an integer to an *_sfract*/*_fract* or visa versa is of very limited use, as there are only two numbers that are both an integer and an *_sfract*/*_fract*: 0 and -1. These conversions are most likely the result of a programming error, so the compiler will flag them as an error.

Conversions of integers to/from the *_accum* type may be useful and are supported.



All three fractional data types can be converted to/from the types `float` and `double`.

3.4.3 PROMOTION RULES

For the three fractional types `_sfract`, `_fract` and `_accum`, the promotion rules are similar to the promotion rules for `char`, `short`, `int` and `long`. This means that for an operation on two different fractional types, the smaller type will be promoted to the larger type before the operation is performed. This larger type will also be the type of the result.

When a fractional type is mixed with a `float` or `double` type, the fractional number is first promoted to `float` respectively `double`. The latter type will also be the type of the result.

When an integer type is mixed with the `_accum` type, the integer is first promoted to `_accum`. The result of the operation will also be of type `_accum`.

Because of the limited range of `_sfract` and `_fract`, only a few operations make sense when combining an integer with an `_sfract` or `_fract`. For that reason, only the following operations are supported in this case:

left	oper	right	result
fractional	*	integer	fractional
integer	*	fractional	fractional
fractional	/	integer	fractional
integer	/	fractional	integer
fractional	<<	integer	fractional
fractional	>>	integer	fractional
fractional: <code>_sfract</code> , <code>_fract</code> integer: <code>char</code> , <code>short</code> , <code>int</code> , <code>long</code>			

Table 3-3: Fractional operations

3.4.4 INTRINSIC FUNCTIONS

A number of intrinsic functions are defined for the fractional data types. All these functions except for the first one are not strictly necessary, because the compiler supports these operations directly. The functions are provided so that you can write more portable code.

The following intrinsic function is defined for the case where you are interested in the integer part of the multiplication of a `_fract` with an integer:

```
long      _mulfractlong ( _fract, long );
```

The following intrinsic can be used to convert a 32 bit fractional data type to a 16 bit fractional data type by rounding the value instead of truncating it. When a `_sat` qualified argument is passed (see next section) then rounding will be done with saturation:

```
_sfract   _round16 ( _fract );
```

The following intrinsic can be used to convert an `_accum` value to a `_fract`:

```
_fract    _getfract ( _accum );
```

The following intrinsic can be used to count the number of consecutive bits which have the same value as bit 15 of an `_sfract`:

```
short     _clssf ( _sfract );
```

These intrinsics can be used to perform a left or right shift on one of the fractional data types. A negative second operand performs a right shift:

```
_sfract   _shasfracts ( _sfract, int );  
_fract    _shafracts ( _fract, int );  
_accum    _shaaccum ( _accum, int );
```

3.5 TYPE QUALIFIER `_SAT`

When a variable is declared with the `_sat` type qualifier, all operations on that variable will be performed using saturating arithmetic. When an operation is performed on a plain variable and a `_sat` variable, the `_sat` takes precedence, and the operation is done using saturating arithmetic. The type of the result of such an operation also includes the `_sat` qualifier, so that another operation on the result will also be saturated. In this respect, the behavior of the `_sat` type qualifier is comparable to the `unsigned` keyword. You can overrule this behavior by inserting type casts with or without the `_sat` type qualifier in an expression.

The `_sat` type qualifier can be used on both fractional and integer types. It is advisable to pay extra attention when using the `_sat` type qualifier on either (un)signed shorts or (un)signed characters, since these types will in many cases be promoted to signed integers before the actual operation takes place, this in accordance with the principles of *usual arithmetic conversions* and *integral promotions*. See also: *ANSI C Type Conversions* in this chapter. The result will then be stored in a saturated way, if the result should be of a saturated small integer type.

Care should also be taken when combining signed and unsigned types, since no saturation between signed and unsigned is done.

Examples:

```
_sat int si          = 0x7FFFFFFF;

int i               = 0x12345;

unsigned int ui     = 0xFFFFFFFF;

si + i // a saturated addition is performed,
       yielding a saturated int

si + ui // a saturated unsigned addition is performed,
        // yielding a saturated unsigned int

i + ui // a normal unsigned addition is performed,
        // yielding an unsigned int
```

3.6 PACKED DATA TYPES

3.6.1 ADDITIONAL BASIC TYPES

Two additional basic types were added to the C compiler to support the packed arithmetic instructions of the TriCore. These types are:

_packb

A 32 bit word consisting of 4 bytes.

_packhw

A 32 bit word consisting of 2 halfwords.

A number of arithmetic operations on packed data types are directly supported by the TriCore instruction set. For example, the following function:

```
_packb add4 ( _packb a, _packb b )
{
    return a + b;
}
```

is translated into the following assembly code:

```
add4:
    add.b    d2,d4,d5
    ret16
```

3.6.2 INTRINSIC FUNCTIONS

The following intrinsic functions let you manipulate packed data types in a way that conforms to the standard C syntax. Code using these intrinsics is portable to another platform, provided that you implement the intrinsics with a library or with macro definitions.

Type conversion of a long integer value to a `_packb` or `_packhw` data type:

```
_packb    _initpackbl    ( long );
_packhw    _initpackhwl    ( long );
```

Build a `_packb/_packhw` value from four/two separate values:

```
_packb    _initpackb      ( int, int, int, int );
_packhw    _initpackhw    ( int, int );
```

Extract an individual byte/halfword from a `_packb/_packhw` value:

```
char       _extractbyte1  ( _packb );
char       _extractbyte2  ( _packb );
char       _extractbyte3  ( _packb );
char       _extractbyte4  ( _packb );
short      _extracthw1    ( _packhw );
short      _extracthw2    ( _packhw );

char       _getbyte1      ( _packb * );
char       _getbyte2      ( _packb * );
char       _getbyte3      ( _packb * );
char       _getbyte4      ( _packb * );
short int  _gethw1        ( _packhw * );
short int  _gethw2        ( _packhw * );
```

Insert one byte/halfword into a `_packb/_packhw` value:

```
_packb     _insertbyte1   ( _packb, char );
_packb     _insertbyte2   ( _packb, char );
_packb     _insertbyte3   ( _packb, char );
_packb     _insertbyte4   ( _packb, char );
_packhw    _inserthw1     ( _packhw, short );
_packhw    _inserthw2     ( _packhw, short );

_packb     _setbyte1      ( _packb *, char );
_packb     _setbyte2      ( _packb *, char );
_packb     _setbyte3      ( _packb *, char );
_packb     _setbyte4      ( _packb *, char );
_packhw    _sethw1        ( _packhw *, short );
_packhw    _sethw2        ( _packhw *, short );
```

Mix four halfwords or 8 bytes into a double register that is represented by the additional datatype `_packw`. To access the values in a `_packw` variable, you can use a union data type: `typedef double _packw`.

```
_packw     _transpose_hword( _packhw, _packhw );
_packw     _transpose_byte ( _packb, _packb );
```

The next intrinsic exchanges the values of *value* and *memory*, but only those bits that are allowed by *mask*. Before the `_swapmsk` instruction is generated, the parameters *value* and *mask* are moved into a double register.

```
_void      _swapmsk      ( int value, int mask, int * memory );
```

Another set of intrinsics is available to give you access to a number of specific TriCore instructions that operate on packed data.

Calculate the absolute value of a `_packb/_packhw` value:

```
_packb      _absb          ( _packb );
_packhw      _absh          ( _packhw );
```

Calculate the absolute value of a `_packhw` value, using saturating arithmetic:

```
_sat _packhw  _abssh      ( _sat _packhw );
```

Calculate the minimum/maximum value for either signed or unsigned `_packb/_packhw` values:

```
_packb      _minb      ( _packb, _packb );
unsigned _packb  _minbu ( unsigned _packb, unsigned _packb );
_packhw      _minh      ( _packhw, _packhw );
unsigned _packhw  _minhu ( unsigned _packhw, unsigned _packhw );
```

3.6.3 HALFWORD PACKED UNIONS AND STRUCTURES

To minimize space consumed by alignment padding with unions and structures, elements follow the minimum alignment requirements imposed by the architecture. The TriCore architecture supports access to 32-bit integer variables on halfword boundaries.

Because only doubles, circular buffers, `_accum` or pointers require the full word access, structures that do not contain members of these types are automatically halfword (2-bytes) packed.

Structures and unions that are divisible by 64-bit or contain members that are divisible by 64-bit, are word packed to allow efficient access through LD.D and ST.D instructions. These load and store operations require word aligned structures that are divisible by 64-bit. If necessary, 64-bit divisible structure elements are aligned or padded to make the structure 64-bit accessible.

With the **pack 2** pragma the "LD.D/ST.D" structure and union copy optimization can be disabled to ensure halfword structure and union packing when possible. This "limited" halfword packing only supports structures and unions that do not contain double, circular buffer, `_accum` or pointer type members and that are not qualified with **#pragma align** to get an alignment larger than 2-byte.



See also section 4.4, *Pragmas* in Chapter 4, *Compiler Use*.

The alignment of data sections and stack may also affect the alignment of the base address of a halfword packed structure. A halfword packed structure can be aligned on a halfword boundary or larger alignment. When located on the stack or at the beginning of a section, the alignment becomes a word, because of the minimum required alignment of data sections and stack objects. A stack or data section can contain any type of object. To avoid wrong word alignment of objects in the section, the section base is also word aligned.

3.7 BIT DATA TYPES

3.7.1 THE _BIT TYPE

The main use of the `_bit` type is to define and manipulate bit objects in memory. A `_bit` type variable can be handled in the same manner as a variable of type `int`. Only some operations of the `_bit` type are directly supported by the TriCore instruction set.

The following rules apply to `_bit` type variables:

1. A `_bit` type variable is always unsigned.
2. A `_bit` type variable can be exchanged with all other type-variables. The compiler generates the correct conversion.

A `_bit` type variable is like a boolean. Therefore, converting an `int` type variable to a `_bit` type variable does not mean the `_bit` type variable is the least significant bit of the `int` type variable. It is 1 (true) if the `int` type variable is not equal to 0, and 0 (false) if the `int` type variable is 0.

In C:

```
bit_variable = int_variable;
```

can be seen as:

```
bit_variable = int_variable ? 1 : 0;
```

3. Pointer to `_bit` is **not** allowed.
4. The `_bit` type is allowed as a structure member.
5. A `_bit` type variable is allowed as a parameter of a function.
6. A `_bit` type variable is allowed as a return type of a function.
7. A `_bit` typed expression is allowed as switch expression.
8. The `sizeof` of a `_bit` type is 1.
9. Global or static `_bit` type variable can be initialized.
10. A `_bit` type variable can be declared absolute using the `_atbit` attribute. See section 3.2.3, *The _atbit() Attribute* for more details.
11. A `_bit` type variable can be declared volatile.

Promotion Rules

For the `_bit` type, the promotion rules are similar to the promotion rules for `char`, `short`, `int` and `long`.

3.7.2 TYPE QUALIFIER _SFRBIT16 AND _SFRBIT32

The type qualifiers `_sfrbit16` and `_sfrbit32` control the access of (SFR) bit fields. Bit fields qualified with the type qualifiers `_sfrbit16` are only accessed as word or half-word. Bit fields qualified with the type qualifiers `_sfrbit32` are only accessed as word.

To support SFR bit field access, SFR data structures are defined containing SFR bit field members. For example the `ctri/include/regcpu_name.sfr` files contain SFR structure definitions that contain SFR bit field members.

In ANSI C, bit fields must be declared as integer type. However, in implementations compliant with the TriCore Embedded Applications Binary Interface, the alignment requirements they impose as members of unions or structures, are the same as those that would be imposed by the smallest integer-based data types wide enough to hold the fields. Thus:

- fields with a width of 8-bits or less impose only byte alignments

- fields with a width from 9 to 16 bits impose half-word alignment
- fields with a width from 17 to 32 bits impose word alignment.

Byte type bit fields are accessed with byte instructions and half-word type bit fields are accessed with half-word instructions. This works well for SFR's that allow byte access, but some of the TriCore SFR's are restricted to 32-bit or 16-bit access.

When you define an SFR bit field with a size equal or smaller than 8-bit for a 32-bit accessible SFR, this may result in byte or half-word access. Also, when you define an SFR bit field with a size equal or smaller than 16-bit for a 32-bit accessible SFR, this may result in half-word access. But byte and half-word operations are not allowed on 32-bit accessible SFR's. Therefor SFR's that only allow 32-bit access, require type qualification with `_sfrbit32` of its bit field members that are equal or smaller than 16-bit. The `_sfrbit32` type qualifier instructs the compiler to generate word access only for accessing the SFR bit field members.

For example

```
#define BCU_Base 0xf0000200 /* BCU block base address */
typedef volatile union
{
    struct
    {
        unsigned _sfrbit32 BCUSRPN : 8; /* BCU Service Request
                                         Priority Number.    */
        unsigned _sfrbit32 : 2;
        unsigned _sfrbit32 BCUTOS : 2; /* BCU Type-of-Service
                                         Control.          */
        unsigned _sfrbit32 BCUSRE : 1; /* BCU Service Request
                                         Enable Control.    */
        unsigned _sfrbit32 BCUSRR : 1; /* BCU Service Request
                                         Flag.              */
        unsigned _sfrbit32 BCUCLRR : 1; /* BCU Request Clear Bit. */
        unsigned _sfrbit32 BCUSETR : 1; /* BCU Request Set Bit.   */
        unsigned _sfrbit32 : 16;
    } B;

    int I;

} BCU_SRC_type;

#define BCU_SRC (*(BCU_SRC_type*)(BCU_Base + 0xfc))
/* BCU Service Request Node */
```

SFR's that allow only 16-bit or 32-bit access, require type qualification with `_sfrbit16` of its bit field members that are equal or smaller than 8-bit. The `_sfrbit16` type qualifier instructs the compiler to generate half-word or word access only for accessing the SFR bit field members.

The `_sfrbit32` and `_sfrbit16` type qualifiers can only be used for `int` types. For example, an error is generated for `_sfrbit32 char x : 8;`

When the `_sfrbit32` and `_sfrbit16` type qualifiers are used for qualifying other types than a bit field, this is ignored without warning. For example `_sfrbit32 int global;` is equal to `int global;`.

Structures or unions that contain a member qualified with `_sfrbit32`, are zero padded when needed to complete a full word. The structure or union will be word aligned. Structure or unions that contain a member qualified with `_sfrbit16`, are zero padded when needed to complete a half-word.

3.8 PARAMETER PASSING

The parameter registers D4..D7 and A4..A7 are used to pass the initial function arguments. Up to 4 arithmetic types and 4 pointers can be passed this way. A 64-bit argument is passed in an even/odd data register pair. Parameter registers skipped because of alignment for a 64-bit argument are used by subsequent 32-bit arguments. Any remaining function arguments are passed on the stack. Stack arguments are pushed in reversed order, so that the first one is at the lowest address. On function entry, the first stack parameter is at the address (SP+0).

All function arguments passed on the stack are aligned on a multiple of 4 bytes. As a result, the stack offsets for all types except `float` are compatible with the stack offsets used by a function declared without a prototype.

Structures up to eight bytes are passed via a data register or data register pair. Larger structures are passed via the stack.

Arithmetic function results of up to 32 bits are returned in the D2 register. 64-bit arithmetic types are returned in the register pair D2/D3. Pointers are returned in A2, and circular pointers are returned in A2/A3.

When the function return type is a structure, it is copied to a "return area" that is allocated by the caller. The address of this area is passed as an implicit first argument in A4.

3.9 FUNCTION QUALIFIERS

The C compiler supports two so-called function qualifiers, to change the calling convention of a function: `_syscallfunc` and `_stackparm`.

3.9.1 INTERRUPT FUNCTION QUALIFIER

The TriCore C language introduces two new reserved words: `_interrupt` and `_interrupt_fast`, which can be seen as special type qualifiers, only allowed with function declarations. A function can be declared to serve as an interrupt service routine. Interrupt functions cannot return anything and must have a **void** argument type list. For example, in:

```
void _interrupt(vector)
    _isr(void)
{
    ...
};
```

The compiler generates an interrupt service frame for interrupts. The `_interrupt` function qualifier takes one argument, *vector*, that defines the interrupt vector number. The difference between a normal function and an interrupt function is that an interrupt function ends with an RFE instruction instead of a RET, and that the lower context is saved and restored with a pair of SVLCX/RSLCX instructions when one of the lower context registers is used in the interrupt handler.

When the interrupt handler function is defined with the `_interrupt()` qualifier, the compiler will generate an entry for the interrupt vector table. This vector will jump to the interrupt handler function. When the function is defined with the `_interrupt_fast()` qualifier, the interrupt handler is directly placed in the interrupt vector table, thereby eliminating the jump redirection code. This should only be used when the interrupt handler is very small, as there is only 32 bytes of space available in the vector table.

Example of `_interrupt`:

Suppose, you want an interrupt function for a software interrupt, and the vector number is 0x30:

```
int c;

void
_interrupt( 0x30 )
transmit(void)
{
    c = 1;
}
```

3.9.2 TRAP FUNCTION QUALIFIERS

With the `_trap` and `_trap_fast` function qualifier you can declare a function to serve as a trap service routine. Trap functions cannot return anything and must have a void argument type list, except class 6 SYS trap handlers. For example, in:

```
void _trap( 7 ) _trapnmi( void )
{
    int tin;

    #pragma asm( d15 )          /* tin number is implicitly */
    #pragma endasm( tin=d15) /* passed via register d15 */

    switch( tin )
    {
        case 0:
            ....
    }
}
```

The compiler generates a trap service frame for traps. The `_trap` function qualifier takes the *class* argument which defines the interrupt trap vector number. The TriCore architecture specifies eight general classes for traps. Each class has its own trap handler, accessed through a trap vector of 32 bytes per entry, indexed by the hardware-defined trap class number. Within each class, specific traps are distinguished by a *Trap Identification Number* (TIN) that is loaded by hardware into register D15 before the first instruction of the trap handler is executed. The trap handler must test and branch on the value in D15 to reach the sub-handler for a specific TIN.

The difference between a normal function and a trap function is that a trap function ends with an RFE instruction instead of a RET, and that the lower context is saved and restored with a pair of SVLCX/RSLCX instructions if one of the lower context registers is used in the interrupt handler. For class 6 SYS traps the lower context is not saved and restored which allows passing and returning parameters.

When the trap handler function is defined with the `_trap()` qualifier, the compiler generates an entry for the trap vector table. This vector jumps to the trap handler function.

When the function is defined with the `_trap_fast()` qualifier, the interrupt handler is directly placed in the trap vector table while eliminating the jump redirection code. This should only be used when the trap handler is very small because there is only 32 bytes of space available in the vector table. This restriction is not checked by the compiler.

The class 6 SYS trap is raised immediately after execution of the SYSCALL instruction to initiate a system call. The SYS trap can be called by functions that are defined with the `_syscallfunc` qualifier. (See section 3.9.5, *System Call Function Qualifier*).

In contradiction to all other traps the SYS trap can return and pass arguments like `_syscallfunc` qualified functions. Arguments that are passed via the stack, remain on the stack of the caller because it is not possible to pass arguments from the user stack to the interrupt stack on a system call. This restriction, caused by the TriCore's run-time behavior, can not be checked by the compiler.

Example

```

_syscallfunc(1) int syscall1( int, int );
_syscallfunc(2) int syscall2( int, int );

int x;

void main( void )
{
    x = syscall1(1,2);
    x = syscall2(4,3);
}

int _trap( 6 ) trap6( int a, int b )
{
    int tin;

#pragma asm( dl5 )
#pragma endasm( tin=dl5 )

    switch( tin )
    {
    case 1:
        a += b;
        break;
    case 2:
        a -= b;
        break;
    default:
        break;
    }
    return a;
}

```

3.9.3 ENABLE INTERRUPT/TRAP FUNCTION QUALIFIER

With the `_enable_` function qualifier you can declare an interrupt or trap service routine to enable the interrupts immediately at function entry. When entering an interrupt or trap service routine, the TriCore interrupt system globally disables the interrupts. For example:


```

void _interrupt(1) _enable_isr( void )
{

}

```

The ENABLE instruction is generated as the first instruction in the interrupt or trap service routine. The ENABLE instruction sets the Interrupt Enable bit (ICR.IE) in the Interrupt Control Register.

The ENABLE instruction can also be generated with the `_enable()` intrinsic function, but for this intrinsic function it is not guaranteed that it will be the first instruction that is executed at interrupt service entry. See also section 3.16, *Intrinsic Functions*

3.9.4 BISR INTERRUPT/TRAP FUNCTION QUALIFIER

With the `_bISR_` interrupt/trap function qualifier you can set the current cpu priority number (ICR.CCPN) to a value in the range of 0 to 511 and enable the interrupts for an interrupt or trap service routine. The lower context is also saved at function entry and restored at function exit for `_bISR_` qualified interrupt or trap functions. When entering an interrupt or trap service routine the TriCore interrupt system globally disables the interrupts. For example:

```

#define CCPN 10
void _interrupt(1) _bISR_(CCPN) isr( void )
{

}

```

The BISR instruction is generated as the first instruction in the interrupt or trap service routine. The BISR instruction saves the lower context, sets the Interrupt Enable bit (ICR.IE) in the Interrupt Control Register and sets the CPU priority number (ICR.CCPN) to the specified CCPN value. At function exit the lower context is restored with a RSLCX instruction.

The BISR instruction can also be generated with the `_bISR()` intrinsic function, but for this intrinsic function it is not guaranteed that it will be the first instruction that is executed at interrupt service entry. See also section 3.16, *Intrinsic Functions*

The `_enable_` function qualifier is superfluous for `_bistr_` qualified functions. A warning will be generated and the `_enable_` qualifier will be ignored.

The `_bistr_` function qualifier is illegal for class 6 traps. The SYS trap, class 6, is raised immediately after execution of the SYSCALL instruction, to initiate a system call. In contradiction to all other traps the SYS trap can return and pass arguments, conform `_syscallfunc` qualified functions. Returning values requires that the lower context may not be restored, return values are returned in the lower context registers. Therefore BISR can not be used, because it saves the lower context. Instead the `_enable_` function qualifier can be used to ENABLE the interrupts and `_mtcr()` intrinsic function can be used to set the ICR.CCPN value at the beginning of a class 6 trap.

3.9.5 SYSTEM CALL FUNCTION QUALIFIER

When you call a function declared with the `_syscallfunc` function qualifier, a SYSCALL instruction is generated rather than a function call. The `_syscallfunc` function qualifier can only be used at a function declaration, not at a function definition. The `_syscallfunc` function qualifier takes one constant argument which is used as the operand of the SYSCALL instruction.

Example:

```
_syscallfunc(42) void trap42(int d4, int d5);
```

The function call `trap42(1, 2)` will result in the following code:

```
movl6    d4,#1
movl6    d5,#2
syscall  #42
```

3.9.6 STACK MODEL FUNCTION QUALIFIER

The function qualifier `_stackparm` changes the standard calling convention of a function into a convention where all function arguments are passed via the stack, conforming a so called stack model. This qualifier is only needed for situations where you need to use an indirect call to a function for which you do not have a valid prototype.

The compiler sets the least significant bit of the function pointer when you take the address of a function declared with the `_stackparm` qualifier, so that these function pointers can be identified at run-time. The least significant bit of a function pointer address is ignored by the hardware.

Example:

```
void          plain_func ( int );
void _stackparm stack_func ( int );

void call_indirect ( void (*fp)( int ), int arg )
{
    typedef _stackparm void (*SFP)( int );
    SFP      fp_stack;

    if ( (int) fp & 1 )
    {
        fp_stack = (SFP) fp;
        fp_stack( arg );
    }
    else
    {
        fp( arg );
    }
}

void main ( void )
{
    call_indirect( plain_func, 1 );
    call_indirect( (void(*)) stack_func, 2 );
}
```

3.9.7 FAR FUNCTION STORAGE QUALIFIER

With the `_far` function storage qualifier you can qualify a function to be called indirectly instead of PC relative or absolute. The `_far` function storage qualifier only effects non C function pointer calls; C function pointer calls always use the indirect call operation of the TriCore.

Use the **-indirect** option to enable code generation for indirect function calling.

The TriCore architecture provides three alternative addressing modes for calls and unconditional jumps: PC relative, absolute and register indirect.

- *PC relative addressing* provides a 24-bit, halfword-scaled relative offset, supporting a target address range from -16Mb to +16Mb around the calling point. This is not always sufficient for calls to functions located within the same memory segment, but it is particularly insufficient for calls across physical memory segments.
- *Absolute addressing* provides a two-complement absolute address with four bits of target segment ID and 20 bits of halfword scaled offset within the segment. Via absolute addressing, targets can be reached within any memory segment but only within the first two megabytes of each segment.
- *Register indirect addressing* provides a full 32-bit target address, held in an address register specified in the instruction. Any target address can be reached with this addressing mode. It only requires to load the address register with the target address.

The PC relative and absolute addressing mode only require one word of code. The indirect addressing requires a 2.5 word sequence (extended load address in address register, call indirect address register). Because of the equal instruction size, PC relative addressing and absolute addressing mode can be combined in a generic call or generic unconditional jump.

To allow the compiler to generate efficient calls to external functions whose final segment and offset remain unknown until locate time, a default compilation mode is defined. You can overrule this mode with the `_far` function storage qualifier or the **-indirect** compiler option. The model for the default operation is as follows:

1. The compiler issues a `callg` or `jg` to the external symbol. These instructions imply generic addressing.
2. The assembler generates a relocatable expression for this generic addressing mode that may be resolved by the locator as a PC relative or absolute call or unconditional jump.
3. If the branch address resolves to a location within +/- 16 Mb range of the `callg` or `jg` instruction, the locator resolves it as a PC relative call or unconditional jump.

Otherwise, if the branch address lies within the first two Mb of the segment to which it is mapped, the locator changes the opcode bits that specify the addressing mode of the `callg` or `jg` instruction, changing the mode to absolute addressing, and resolves the branch as an absolute call or unconditional jump.

4. If neither of the above two conditions holds, the locator generates an error telling that the target address is not in range of the generic branch. In this case the function that defines this target address can be qualified with `_far` to solve this problem.

In case you do not want to check your code on target addresses that might be out of range, you can use the **-indirect** compiler option to implicitly qualify all functions to be called indirectly. The (less efficient) 2.5 word indirect call sequences will then be generated.

Naming convention for CODE sections

The CODE sections generated for `_far` qualified functions use a section name `farcode.mod_name` instead of the default code section name `code.mod_name`. This naming convention makes it possible to identify `farcode` and group `farcode` sections together. Within one source module, all `farcode` sections are grouped together. With the **-R** compiler option or with the **#pragma section** all `farcode` sections of your application can be grouped together.



See Section 4.2.1, *Detailed Description of the Compiler Options* for a description of the compiler option **-R**.

See section 4.4, *Pragmas* for a description of **#pragma section**.

The C libraries conform to the default compilation model. To call the C library functions indirectly, use the **-indirect** option or use C function pointer calls. C library functions should not be qualified `_far` and the run-time library functions, mostly written in assembly, cannot be qualified `_far`. To use indirect calling, you do not have to compile the C library functions because most library functions are leaf functions or call other library functions. The library functions linked with your application are located in one section per library, PC relative addressing would be sufficient for internal library calls.

3.10 TYPE QUALIFIER VOLATILE

You can use the `volatile` type qualifier when modifications on the object have undesired side effects when they are performed in the regular way. Memory locations may not be updated because of compiler optimizations, which attempt to save a memory write by keeping the value in a register. When a variable is declared with the `volatile` qualifier, the compiler disables such optimizations.

The ANSI report describes that the updates of volatile objects follow the rules of the abstract machine (the target processor) and thus access to a volatile object becomes implementation defined.

Example:

```
const volatile _near int real_time_clock _at(0x1234);

/*  define the real time clock register;
    it is read-only (const);
    read operations must access the real memory
    location (volatile)
*/
```

3.11 TYPE QUALIFIERS RESTRICT AND _RESTRICT

You can apply the `restrict` or `_restrict` type qualifier to pointer types that point to an object and not a function. An object that is accessed through a `restrict`-qualified pointer has a special association with that pointer. This association requires that all accesses to that object use, directly or indirectly, the value of that particular pointer. The intended use of the `restrict` qualifier is to promote optimization, and deleting all instances of the qualifier from a program does not change the meaning of the program.

Only the `restrict` keyword can be enabled or disabled by the **-Ar** (default) or **-AR** option respectively.

The `restrict` keyword is described in detail in the ISO/IEC 9899:1999(E) standard, Programming languages – C.

Example:

```
_sfract * restrict a;
_sfract * restrict b;
```

declares two `restrict` qualified pointers to an `_sfract` object. If an object is accessed using one of `a` or `b`, and that object is modified anywhere in the program, then it is never accessed using the other one.

3.12 STRINGS

In this section the word 'strings' means the separate occurrence of a string in a C program. So, array variables initialized with strings are just initialized character arrays, which can be allocated in any memory type, and are not considered as 'strings'.

Strings and literals in a C source program, which are not used to initialize an array, have static storage duration. The ANSI X3.159-1989 standard permits string literals to be put in ROM. You can instruct the compiler to allocate strings in ROM (CODE) memory with the **-c** option. By default, **ctri** allocates strings in DATA memory. Note that initialized arrays are still located in RAM.

```
char ramhelp[] = "help";
/* allocation of 5 bytes in RAM and
   5 bytes in ROM */
```

Example of an array in ROM only, initialized with the addresses of strings, also ROM only:

```
char * message[] = {"hello", "alarm", "exit"};
```

ANSI string concatenation is supported: adjacent strings are concatenated – only when they appear as primary expressions – to a single new one. The result may not be longer than the maximum string length (ANSI limit 509 characters, actual compiler limit 1500 characters).

The ANSI Standard states that identical string literals need not be distinct, i.e. may share the same memory. Because memory can be very scarce with microcontroller applications, the **ctri** compiler overlays identical strings within the same module.

In section 3.1.4 the Standard states that behavior is undefined if a program attempts to modify a string literal. Because it is a common extension to ANSI (A.6.5.5) that string literals are modifiable, there may be existing C source modifying strings at run-time. This can be done with pointers, or even worse:

```
"string"[2] = 'r';
```



Note that identical strings are overlaid!

3.13 VARIABLE ARGUMENT LISTS

A special function call convention is used when a function is declared to accept a variable number of arguments. The difference with the normal parameter passing convention is that all variable arguments are always passed on the stack. Default argument promotion takes place for these arguments. The variable arguments can be accessed with the ANSI C macros defined in `stdarg.h`.

3.14 INLINE C FUNCTIONS

The `_inline` keyword is used to signal the compiler to inline the function body instead of calling the function. An inline function must be defined in the same source file before it is 'called'. When an inline function has to be called in several source files, each file must include the definition of the inline function. Usually this is done by defining the inline function in a header file.

Not using a function which is defined as an `_inline` function does not produce any code.

Example (t.c):

```
int  w,x,y,z;

_inline int
add( int a, int b )
{
    return( a + b );
}

void
main( void )
{
    w = add( 1, 2 );
    z = add( x, y );
}
```

No specific debug information is generated about inline functions. The debugger cannot step-into an inline function, it considers the inline function as one HLL source line.

The pragmas `asm` and `endasm` are allowed in inline functions. This makes it possible to define inline assembly functions. See also the section *Inline Assembly* in this chapter.

The generated code is:

```
main:
    movl6    d15,#3
    st.w     w,d15
    ld.w     d15,y
    ld.w     d7,x
    addl6    d15,d7
    st.w     z,d15
    retl6
```

3.15 INLINE ASSEMBLY

ctri supports inline assembly using the following pragmas:

#pragma asm Insert assembly text following this pragma.

#pragma asm_noflush As **#pragma asm**, but without flushing optimizer information.

#pragma endasm Switch back to the C language.



C modules containing inline assembly are not portable and are very hard to prototype in other environments.

When the compiler encounters a pragma **asm**, it discards some information gathered for optimization. This means that optimizations like copy and constant propagation are not performed across inline assembly code. If you use pragma **asm_noflush** instead, these optimizations are not disabled.

C Variable Interface

The pragmas can be followed by a specification of the register interface between the C code and the inline assembly code. After the **#pragma asm** or **asm_noflush**, you can allocate fixed registers or scratch registers that can be used in the assembly code. These registers may optionally be initialized with the value of a C variable. The compiler makes sure that C variables that are needed after the inline assembly fragment are not allocated in those registers.

After the **#pragma endasm**, you can add assignments from a register back to a C variable. All registers mentioned here, should have been allocated in the corresponding **#pragma asm**.

The syntax of the pragmas is as follows:

#pragma asm [(*reg*[=*varname*][, *reg*[=*varname*] ...)]

#pragma asm_noflush [(*reg*[=*varname*][, *reg*[=*varname*] ...)]

#pragma endasm [(*varname*=*reg*[, *varname*=*reg*] ...)]

The arguments of the pragmas are:

varname name of a C variable,

reg a fixed register or a scratch register.

You can use the following fixed registers:

d0..d15
a2..a7, a11..a15
e0..e14 (data registers pairs)
c2..c6, c12..c14 (address register pairs)

A scratch register name has the following syntax:

{**d**|**a**|**e**|**c**}%*index*

d data register

a address register

e data registers pair

c address register pair

index is a user defined number in the range 0–9. In the inline assembly code, escape sequences consisting of a percent sign followed by a number are replaced by the corresponding scratch register that was allocated by the compiler. Registers are not replaced inside strings or comments. When the scratch register is a register pair (e%0, c%0), you can substitute the low or high part of the register pair by adding an '**l**' or '**h**' after the '%' sign.

Example:

```
int doit ( int a, int b )
{
#pragma asm ( d0, d1=a, d%2, d%3=b )
    MOV d0,d1
    MOV %2,%3
#pragma endasm ( a=d0, b=d%2 )
    return a + b;
}
```

This example allocates four data registers. The first two, `d0` and `d1`, are real registers, the last two, `d%2` and `d%3` are scratch registers that are allocated automatically. Register `d1` is initialized with the value of variable `a`, and the scratch register `d%3` is initialized with the value of variable `b`. Afterwards, the compiler makes sure that the value of `d0` becomes accessible as the C variable `a`, and scratch register `d%2` becomes available as variable `b`. This will be done by either allocating the C variable in the register, or when this is not possible, by generating a `MOV` instruction.

You can use inline assembly code in a function that is declared `_inline`. This makes it possible to write an `_inline` function that acts like a C wrapper around your inline assembly code.

3.16 INTRINSIC FUNCTIONS

When you want to use specific TriCore instructions that have no equivalence in C, you would be forced to write assembly routines to perform these tasks. However, **ctri** offers a way of handling this in C. **ctri** has a number of built-in functions, which are implemented as intrinsic functions.

To the programmer intrinsic functions appear as normal C functions, but the difference is that they are interpreted by the code generator, so that more efficient code may be generated. Several pre-declared functions are available to generate inline assembly code at the location of the intrinsic function call. This avoids the overhead that is normally introduced by parameter passing and context saving before executing the called function.

The names of the intrinsic functions all have a leading underscore, because the ANSI specification states that public C names starting with an underscore are implementation defined.

The advantages of using intrinsic functions, compared with in-line assembly (pragma asm/endasm) are:

- the possibility to use simulation routines or stub functions by a host compiler, to replace the inline assembly code generated by **ctri**
- C level variables can be accessed
- the compiler chooses to generate the most efficient code to access C variables
- intrinsic code is optimized, except for `_nop()`

The intrinsic `_disable` yields an indication whether the interrupts were enabled before. This indication can be passed to the `_restore` intrinsic to restore the state of the interrupt handling.

```
void    _enable    ( void );
int     _disable   ( void );
void    _restore   ( int );
```

The following intrinsic functions generate the single instruction that corresponds to the function name. The `_syscall` and `_bistr` functions require a constant argument:

void	_debug	(void);
void	_dsync	(void);
void	_isync	(void);
void	_svlcx	(void);
void	_rslcx	(void);
void	_nop	(void);
void	_syscall	(int);
void	_bistr	(int);

These intrinsics can be used to access control registers with the MFCR and MTCR instructions. The first argument is the constant 16-bit register offset:

int	_mfcrr	(int);
void	_mtcrr	(int, int);

A number of specialized instructions that operate on a register value and return a value in another register, can be accessed via the following intrinsic functions:

int	_clz	(int);
int	_clo	(int);
int	_cls	(int);
int	_satb	(int);
int	_satbu	(int);
int	_sath	(int);
int	_sathu	(int);
int	_abs	(int);
int	_abss	(int);
int	_parity	(int);

To determine the minimum or maximum value of a signed or unsigned value with a single instruction, you can use the following intrinsics:

int	_min	(int, int);
short	_mins	(short, short);
unsigned	_minu	(unsigned, unsigned);
int	_max	(int, int);
short	_maxs	(short, short);
unsigned	_maxu	(unsigned, unsigned);

Extracting a bit field from an integer with the EXTR or EXTRU instruction can be done with the `_extr()` or `_extru()` intrinsics. Inserting a bit field with the INSERT instruction is done with the `_insert()` intrinsic. The `pos` and `width` arguments must be constant:

```
int          _extr      ( int value, int pos, int width );
unsigned     _extru     ( int value, int pos, int width );
int          _insert    ( int old, int new, int pos, int width );
```

Access to the INS and INSN instruction is provided through the `_ins()` and `_insn()` intrinsics. The `oldbit` and `newbit` arguments specify the bit positions in the original value and new value. These bit positions must be constants:

```
int          _ins       ( int old, int oldbit, int new, int newbit );
int          _insn      ( int old, int oldbit, int new, int newbit );
```

To multiply two 32-bit number to a 64-bit result, and scale back the result to a 32-bit result, you can use the `_mulsc()` intrinsic. The third operand specifies the number of bits that should be dropped from the result:

```
int          _mulsc     ( int a, int b, int scale );
```

Additional intrinsic functions are available for manipulating fractional numbers, packed data types and circular pointers. Refer to the description of these language extensions for a description of the associated intrinsic functions.

When you invoke **ctri** with the command line option **-builtin**, a list of prototypes for all intrinsic functions is displayed.

With the **_imaskldmst()** intrinsic function you can perform atomic Load-Modify-Store of a bit-field from an integer value with the IMASK and LDMST instruction. This intrinsic writes the number of *bits* of an integer *value* at a certain *address* location in memory with a *bitoffset*. The number of *bits* must be a constant value:

```
void         _imaskldmst ( int* address, int value, int bitoffset, int bits )
```

With the intrinsic macro **_putbit()** you can store a single bit atomically in memory at a specified bit offset. The bit at offset 0 in *value* is stored at an *address* location in memory with a *bitoffset*:

```
void         _putbit     ( int value, int* address, int bitoffset )
```

This intrinsic is implemented as a macro definition which uses the **_imaskldmst()** intrinsic:

```
#define _putbit ( value, address, bitoffset ) _imaskldmst ( address,  
value, bitoffset, 1 )
```

With the intrinsic macro **_getbit()** you can load a single bit from memory at a specified bit offset. A bit value is loaded from an *address* location in memory with a *bitoffset* and returned as an unsigned integer value:

```
unsigned _getbit ( int* address , int bitoffset );
```

This intrinsic is implemented as a macro definition which uses the **_extru()** intrinsic:

```
#define _getbit ( address, bitoffset ) _extru ( *(address), bitoffset, 1 )
```


3.17 MISRA C

Based upon the 'MISRA guidelines for the application of C language in vehicle based software', the TASKING MISRA C technology offers enhanced compiler error checking that will guide the programmer in writing better, more coherent and intrinsically safer applications. Through this configurable system of enhanced C language error checking, the use of error-prone C constructs can be prevented. A predefined configuration for compliance with the 'required rules' described in the MISRA guidelines is selectable through a single click in the EDE|MISRA C Compiler Options menu. A custom set of applicable MISRA C rules can be easily configured using the same menu. It is also possible to have a project team work with a MISRA C configuration common to the whole project. In this case the MISRA C configuration can be read from an external settings file. This too, is easily selected through the EDE|MISRA C Compiler Options menu. In order to provide proof that installed company MISRA C requirements have in fact been adhered to throughout the entire project, the TriCore Linker/Locator can generate a MISRA C Quality Assurance report. This report lists the various modules in the project with the respective MISRA C settings under which these have been compiled.

Unfortunately it has not been possible to implement support for all 127 rules described in the MISRA guidelines. The reason for this is that a number of rules are beyond the scope of what can be checked in a C compiler environment. These unsupported rules are visible in the EDE|MISRA C Compiler Options menu dialog boxes, but cannot be selected (grayed out).

MISRA is a registered trademark of MIRA held on behalf of the Motor Industry Software Reliability Association.

Enabling MISRA C

From the command line MISRA C can be enabled by the following compiler option:

```
-misracn,n,...
```

where *n* specifies the rule(s) which must be checked.

Error Messages

In case a MISRA C rule is violated, an error message will be generated e.g.:

E 209: MISRA C rule 9 violation: comments shall not be nested.

See Appendix B *MISRA C* for the supported and unsupported MISRA C rules.

3.18 STRUCTURE TAGS

A tag declaration is intended to specify the lay-out of a structure or union. If a memory type is specified, it is considered to be part of the declarator. A tag name itself, nor its members can be bound to any storage area, although members having type "... pointer to" do require one. A tag may then be used to declare objects of that type, and may allocate them in different memories (if that declaration is in the same scope). The following example illustrates this constraint.

```
struct S {
    _near int i;          /* referring to storage: not correct */
    _far char *p;         /* used to specify target memory: correct */
};
```

In the example above **ctri** ignores the erroneous `_near` storage qualifier (without displaying a warning message).

3.19 TYPEDEF

Typedef declarations follow the same scope rules as any declared object. Typedef names may be (re-)declared in inner blocks but not at the parameter level. However, in typedef declarations, memory specifiers are allowed. A typedef declaration should at least contain one type specifier.

Examples:

```
typedef _near int NEARINT;      /* storage type _near: OK */
typedef int _far *PTR;         /* logical type _far
                               storage type 'default' */
```

3.20 CIRCULAR BUFFERS

The TriCore provides a specific addressing mode that can be used to efficiently implement a circular buffer. This type of buffer is often used in DSP applications. The C compiler **ctri** supports this addressing mode with the pointer qualifier **_circ**. When you define a pointer variable with this keyword, operations on this pointer will be performed using the circular addressing mode. Because a circular pointer consists of two address registers, it is eight bytes in size.

The following example shows how a circular pointer is defined:

```
_fract _circ buffer[SIZE];  
_fract _circ *circBuffer = buffer;
```

If you want to initialize a circular pointer with a dynamically allocated buffer at run-time, you should use the intrinsic function **_initcirc()**:

```
#define N 100  
_fract *buf = calloc( N, sizeof(_fract) );  
circBuffer = _initcirc( buf, N, 0 );
```

3.21 SWITCH STATEMENT

ctri supports three ways of code generation for a switch statement: a jump chain (linear switch), a jump table or a lookup table.

A jump chain is comparable with an if/else-if/else-if/else construction. A jump table is a table filled with target addresses for each possible switch value. The switch argument is used as an index within this table. A lookup table is a table filled with a value to compare the switch argument with and a target address to jump to. The compiler generates a call to a run-time library function, which performs a binary search lookup.

By default, the compiler will automatically choose the most efficient switch implementation based on code and data size and execution speed. The selection of the switch method can be influenced by the **-Os** option, which favors execution speed over code size.

It is obvious that, especially for large switch statements, the jump table approach executes faster than the lookup table approach. Also the jump table has a predictable behavior in execution speed. No matter the switch argument, every case is reached in the same execution time. However, when the case labels are distributed far apart, the jump table becomes sparse, wasting code memory. The compiler will not use the jump table method when the waste becomes excessive.

With a small number of cases, the jump chain method can be faster in execution and shorter in size.

The compiler chosen switch method can be overruled by using:

```
#pragma switch linear      /* force jump chain code */
#pragma switch jumptab    /* force jump table code */
#pragma switch lookup     /* force lookup table
                           code */
#pragma switch auto        /* let the compiler decide
                           the switch method used */
#pragma switch restore    /* restore previous switch
                           method */
```

`Pragma switch auto` is also the default of the compiler.



It is not possible to change the switch method selection inside a function definition. Therefore, you should only use `#pragma switch` at file scope.

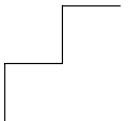
CHAPTER

4

COMPILER USE



TASKING



4

CHAPTER

4.1 CONTROL PROGRAM

The control program **cctri** facilitates the invocation of the various components of the TriCore toolchain, from a single command line. The control program accepts source files and options on the command line in random order.

The invocation syntax of the control program is:

```
cctri [ option ] ... [ control ] ... [ file ] ... ] ...
```

Options are preceded by a '-' (minus sign). The input *file* can have one of the extensions explained below.



When you use a **UNIX** shell (Bourne shell, C-shell), arguments containing special characters (such as '(' and '?') must be enclosed with " " or escaped. The -? option (in the C-shell) becomes: "-?" or -\?.

The control program recognizes the following argument types:

- Arguments starting with a '-' character are options. Some options are interpreted by the control program itself; the remaining options are passed to those programs in the tool chain that accept the option.
- Arguments with a .c suffix are interpreted as C source programs and are passed to the compiler.
- Arguments with a .asm or .src suffix are interpreted as assembly source files. They are directly passed to the assembler.
- Arguments with a .a suffix are interpreted as library files and are passed to the linker.
- Arguments with a .obj suffix are interpreted as object files and are passed to the linker.
- Arguments with a .out suffix are interpreted as linked object files and are passed to the locator. The locator accepts only one .out file in the invocation.
- Arguments with a .dsc suffix are treated as locator command files. If there is a file with extension .dsc on the command line, the control program assumes a locate phase has to be added. If there is no file with extension .dsc, the control program stops after linking (unless it has been directed to stop in an earlier phase)
- Everything else is considered an object file and is passed to the linker.

Normally, a control program tries to compile and assemble all source files to object files, followed by a link and locate phase which produces an absolute output file. There are however, options to suppress the assembler, linker or locator stage. The control program produces unique filenames for intermediate steps in the compilation process, which are removed afterwards.

The following options are interpreted by the control program:

Option	Description
-? or none	Display invocation syntax
-Waarg	Pass argument directly to the assembler
-Wcarg	Pass argument directly to the C compiler
-Wcparg	Pass argument directly to the C++ compiler
-Wlkarg	Pass argument directly to the linker
-Wlcarg	Pass argument directly to the locator
-Wplarg	Pass argument directly to the C++ pre-linker
-V	Display version header only
-c++	Force .c files to C++ mode
-c	Do not link: stop at .obj
-cc	Compile C++ files to .c and stop
-cl	Do not locate: stop at .out
-cm	Always also invokes the C++ muncher
-cp	Always also invokes the C++ pre-linker
-cs	Do not assemble: compile C files to .src and stop
-elf	Set locator output file format to ELF/DWARF
-f file	Read arguments from <i>file</i> ("-" denotes standard input)
-fptrap	Use a floating point library with trap handler.
-ieee	Set locator output file format to IEEE-695 (default)
-ihex	Set locator output file format to Intel Hex
-nolib	Do not link with the standard libraries
-o file	Specify the output file
-srec	Set locator output file format to Motorola S-records
-tiof	Set locator output file format to TIOF-695
-tmp	Keep intermediate files
-v	Show command invocations

Option	Description
-v0	Show command invocations, but do not start them
-wc++	Enable C and assembler warnings for C++ files

Table 4-1: Control program options



For more detailed information about the control program **cctri**, refer to section *cctri* in Chapter *Utils* of the *Cross-Assembler Linker/Locator, Utilities User Guide*.

4.2 COMPILER

The invocation syntax of the C compiler is:

```
ctri [ [option] ... [file] ... ] ...
```



When you use a **UNIX** shell (Bourne shell, C-shell), arguments containing special characters (such as '(' and '?') must be enclosed with " " or escaped. The **-?** option (in the C-shell) becomes: **"-?"** or **-\?**.

The C compiler accepts C source file names and command line options in random order. Source files are processed in the same order as they appear on the command line (left-to-right). Options are indicated by a leading '-' character. Each C source file is compiled separately and the compiler generates an output file with suffix **.src** per C source module, containing assembly source code.

The priority of the options is left-to-right: when two options conflict, the first (most left) one takes effect. The **-D** and **-U** options are not considered conflicting options, so they are processed left-to-right for each source file. You can overrule the default output file name with the **-o** option. The compiler uses each **-o** option only once, so it is possible to specify multiple **-o** options for multiple source files.

When you invoke **ctri** without any argument, the invocation syntax is displayed (same as **-?** option).

A summary of the options is given below. The next section describes the options in more detail.

Option	Description
-?	Display invocation syntax
-A[<i>flag...</i>]	Control language extensions
-Ccpu	Include SFR definition file <i>regcpu.sfr</i> before source
-Dmacro[=<i>def</i>]	Define preprocessor <i>macro</i>
-E[m l]	Preprocess only or emit dependencies or enable multi-line macros
-F	Treat 'double' as 'float'
-Fc	Enable 'float' constants
-FPU	Allow the use of single precision floating point hardware instructions
-Hfile	Include <i>file</i> before starting compilation

Option	Description
-Idirectory	Look in <i>directory</i> for include files
-K	No type-checking for old-style (K&R) function calls
-N	Allocate all data in <code>_near</code> memory
-Nthreshold	Threshold for <code>_near/_far</code> allocation (default 8)
-Oflag...	Control optimization
-R[name]	Change default section name
-TC2	Allow use of TriCore2 instructions
-Umacro	Remove preprocessor <i>macro</i>
-V	Display version header only
-WAE	Treat warning messages as errors
-Z	Allocate all data in <code>_a0</code> memory
-Zthreshold	Threshold for <code>_a0</code> allocation (default 0)
-builtin	Display the list of builtin (intrinsic) functions
-c	Allocate strings in code memory
-e	Remove output file if compiler errors occur
-err	Send diagnostics to error list file (<code>.err</code>)
-f file	Read options from <i>file</i>
-g[f l n]...	Enable symbolic debug information (unless -gn is used)
-indirect	Generate indirect call and jump instructions for indirect function calling
-misracn,n,...	Enable individual MISRA C checks
-n	Send output to standard output
-o file	Specify name of output <i>file</i>
-s	Merge C-source code with assembly output
-u	Treat all 'char' variables as unsigned
-w[num]	Suppress one or all warning messages
-wstrict	Suppress strict warnings (196,303)
-zpragma	Identical to '#pragma <i>pragma</i> ' in the C source

Table 4-2: Compiler options (alphabetical)

Description	Options
Include options	
Read options from <i>file</i>	-f <i>file</i>
Include SFR definition file <i>regcpu.sfr</i> before source	-C <i>cpu</i>
Include <i>file</i> before starting compilation	-H <i>file</i>
Look in <i>directory</i> for include files	-Id <i>directory</i>
Preprocess options	
Preprocess only or emit dependencies or enable multi-line macros	-E[m l]
Define preprocessor <i>macro</i>	-D <i>macro</i> [= <i>def</i>]
Remove preprocessor <i>macro</i>	-U <i>macro</i>
Allocation control options	
Allocate all data in <i>_near</i> memory	-N
Threshold for <i>_near/_far</i> allocation (default 8)	-N <i>threshold</i>
Allocate all data in <i>_a0</i> memory	-Z
Threshold for <i>_a0</i> allocation (default 0)	-Z <i>threshold</i>
Change default section name	-R[<i>name</i>]
Allocate strings in code memory	-c
Code generation options	
Allow the use of single precision floating point hardware instructions	-FPU
Control optimization	-O <i>flag</i> ...
Allow use of TriCore2 instructions	-TC2
Generate indirect call and jump instructions for indirect function calling	-indirect
Identical to '#pragma <i>pragma</i> ' in the C source	-z <i>pragma</i>
Language control options	
Enable/disable specific language extensions	-A[<i>flag</i> ...]
Treat 'double' as 'float'	-F
Enable 'float' constants	-Fc
No type-checking for old-style (K&R) function calls	-K
Treat all 'char' variables as unsigned	-u

Description	Options
Output file options	
Display the list of builtin (intrinsic) functions	-builtin
Remove output file if compiler errors occur	-e
Send output to standard output	-n
Specify name of output <i>file</i>	-o file
Merge C-source code with assembly output	-s
Diagnostic options	
Display invocation syntax	-?
Display version header only	-V
Treat warning messages as errors	-WAE
Send diagnostics to error list file (<code>.err</code>)	-err
Enable symbolic debug information (unless -gn is used)	-g[f l n]...
Enable individual MISRA C checks	-misracn,n,\dots
Suppress one or all warning messages	-w[num]
Suppress strict warnings (196,303)	-wstrict

Table 4-3: Compiler options (functional)

4.2.1 DETAILED DESCRIPTION OF THE COMPILER OPTIONS

Option letters are listed below. Each option (except **-o**; see description of the **-o** option) is applied to every source file. If the same option is used more than once, the first (most left) occurrence is used. The placement of command line options is of no importance except for the **-I** and **-o** options. For the **-o** option, the filename may not start immediately after the option. There must be a tab or space in between. All other option arguments must start immediately after the option. Source files are processed in the same order as they appear on the command line (left-to-right).

Some options have an equivalent pragma.



With options that can be set from within EDE, you will find a mouse icon that describes the corresponding action.

**Option:**

-?

Description:

Display an explanation of options at stdout.

Example:

```
ctri -?
```


-A

Option:



Select the Project | C Compiler Options | Project Options... menu item. Set or disable the Language Extensions in the Language tab.



-A[*flags*]

Arguments:

Optionally one or more language extension flags.

Default:

-A1

Description:

Control language extensions. **-A** without any flags, specifies strict ANSI mode; all language extensions are disabled. This is equivalent to **-ACKLPRSTV** and **-A0**.

Flags which are controlled by a letter, can be switched on with the lower case letter and switched off with the uppercase letter. Note that the usage of these options might have effect on code density and code execution performance. The following flags are allowed:

- c** Default. Do not check for assignments of a constant string to a non-constant string pointer. With this option the following example produces no warning:

```
char *p;
void main( void ) { p = "hello"; }
```

- C** Conform to ANSI-C by checking for assignments of a constant string to a non-constant string pointer. The example above produces warning W130: "operands of '=' are pointers to different types".

- k** Default. Allow keyword language extensions such as `_near`, `_far` and `_at`.

- K** Keyword extensions are not allowed.

- l** Default. 500 significant characters are allowed in an identifier instead of the minimum ANSI-C translation limit of 31 significant characters. Note: more significant characters are truncated without any notice.
- L** Conform to the minimum ANSI-C translation limit of 31 significant characters. This makes it possible to translate your code with any ANSI-C conforming C-compiler. Note: more significant characters are truncated without any notice.
- p** Default. Allow C++ style comments in C source code. For example:

```
// e.g this is a C++ comment line.
```
- P** Do not allow C++ style comments in C source code, to conform to strict ANSI-C.
- r** Default. Allow the use of the `restrict` qualifier.
- R** Do not allow the use of the `restrict` qualifier.
- s** Default. `__STDC__` is defined as '0'. The decimal constant '0', intended to indicate a non-conforming implementation. When one of the language extensions are enabled `__STDC__` should be defined as '0'.
- S** `__STDC__` is defined as '1'. In strict ANSI-C mode (**-A**) `__STDC__` is defined as '1'.
- t** Default. Do not promote old-style function parameters when prototype checking.
- T** Perform default argument promotions on old-style function parameters for a strict ANSI-C implementation. `char` type arguments are promoted to `int` type and `float` type arguments are then promoted to `double` type.
- v** Default. Allow type cast of an lvalue object with incomplete type `void` and lvalue cast which does not change the type and memory of an lvalue object.

Example:

```
void *p; ((int*)p)++;      /* allowed */
int i; (char)i=2;         /* NOT allowed */
```

- V** A cast may not yield an lvalue, to conform strict ANSI-C mode.

0 – same as **-ACKLPRSTV** (disable all, strict ANSI-C)

1 – same as **-Acklprstv** (default, enable all)

Example:

To disable language keyword extensions and C++ comments enter:

```
ctri -AKP test.c
```

-builtin

Option:

-builtin

Description:

Displays a list of function prototypes for all intrinsic functions.

Example:

```
ctri -builtin test.c
```

-C

Option:



Select the **Project | Processor Options...** menu item and choose the **CPU** tab. Select a CPU type in the **CPU type** field.

If you select **User** defined in the **CPU type** field, type the name of your TriCore derivative in the **User specified CPU name** field.

If you select **User** defined in the **CPU type** field and leave the **User specified CPU name** field empty, the option **-C** is not used.



-Ccpu

Arguments:

The CPU name which identifies your TriCore derivative.

Description:

Use special function register definitions for *cpu*. The filename looked for is "reg*cpu*.sfr". The search algorithm for .sfr files is the same as for include files that are enclosed in "" at the beginning of the C source. The file is included before the source.

Example:

To use SFR definitions from the file `regtc10gp.sfr`, enter:

```
ctri -Ctc10gp test.c
```

-C

Option:



Select the Project | C Compiler Options | Project Options... menu item. Enable the Allocate strings in code memory check box in the Allocation tab.



-c

Description:

Use this option to force allocation of strings or floating point constants in code memory. By default, these constants are allocated in data memory.

Example:

```
ctri -c test.c
```

-D

Option:



Select the Project | C Compiler Options | Project Options... menu item. Define a macro (syntax: *macro[=def]*) in the Define user macros field in the Preprocessing tab. You can define more macros by separating them with commas.



-Dmacro[=def]

Arguments:

The macro you want to define and optionally its definition.

Description:

Define *macro* to the preprocessor, as in `#define`. If *def* is not given (`=` is absent), `'1'` is assumed. Any number of symbols can be defined. The definition can be tested by the preprocessor with `#if`, `#ifdef` and `#ifndef`, for conditional compilations. If the command line is getting longer than the limit of the operating system used, the **-f** option is needed.

Example:

The following command defines the symbol `NORAM` as `1` and defines the symbol `PI` as `3.1416`.

```
ctri -DNORAM -DPI=3.1416 test.c
```



-U

-E / -Em / -EI

Option:



Select the Project | C Compiler Options | Project Options... menu item. Enable the Preprocess only and capture output check box in the Preprocessing tab.



-E[m | I]

Description:

Run the preprocessor of the compiler only and send the output to stdout. When you use the **-E** option, use the **-o** option to separate the output from the header produced by the compiler.

When you use the **-Em** option, the compiler generates dependency rules which can be used by a 'make' utility.

When you use the **-EI** option, you can use multi-line macros. A backslash used to continue a macro on the next source line will be expanded as a new line instead of a concatenation of the lines.

Examples:

The following command preprocesses the file `test.c` and sends the output to the file `preout`.

```
ctri -E -o preout test.c
```

The following command preprocesses the file `test.c` which may contain multi-line macros, and sends the output to the file `multi`.

```
ctri -EI test.c -o multi
```

The following command generates dependency rules for the file `test.c` which can be used by **mktri** (the TriCore 'make' utility).

```
ctri -Em test.c
```

```
test.src : test.c
```


-e

Option:



EDE always removes the output file on errors.



-e

Description:

Remove the output file when an error has occurred. With this option the 'make' utility always does the proper productions.

Example:

```
ctri -e test.c
```

-err

Option:



In EDE this option is not so useful. If you would use this option you would not see the error messages in the Build tab.



-err

Description:

Write errors to the file *source.err* instead of stderr.

Example:

To write errors to the `test.err` instead of stderr, enter:

```
ctrl -err test.c
```

-F / -Fc

Option:



Select the Project | C Compiler Options | Project Options... menu item. Enable the Treat type 'double' as 'float' and/or Use type 'float' instead of 'double' for floating point constants check box in the Misc tab.



-F[c]

Description:

-F forces using single precision floating point only, even when double or long double is used. In fact double and long double are treated as float and default argument promotion from float to double is suppressed. When you use this option, you must use the single precision version of the C library.



See the chapter *Libraries* for the naming conventions of the standard libraries.

-Fc enables the use of 'float' constants. In ANSI C floating point constants are treated having type double, unless the constant has the suffix **f**. So '3.0' is a double precision constant, while '3.0f' is a single precision constant. This option tells the compiler to treat all floating point constants as single precision float types (unless they have an explicit 'l' suffix).

Example:

To force double to be treated as float, enter:

```
ctri -F test.c
```

-f

Option:

-f *file*

Arguments:

A filename for command line processing. The filename "-" may be used to denote standard input.

Description:

Use *file* for command line processing. To get around the limits on the size of the command line, it is possible to use command files. These command files contain the options that could not be part of the real command line. Command files can also be generated on the fly, for example by the make utility.

More than one **-f** option is allowed.

Some simple rules apply to the format of the command file:

1. It is possible to have multiple arguments on the same line in the command file.
2. To include whitespace in the argument, surround the argument with either single or double quotes.
3. If you use quotes inside a quoted argument, use the following syntax:
 - a. If you use a single quote, surround the argument with double quotes. If you use a double quote, surround the argument with single quotes.

Example:

"Single quote ' embedded"

'Double quote " embedded'

- b. If you use both types of quotes, split the argument in such a way that each embedded quote is surrounded by the opposite type of quote.

'Double quote " and single quote "' embedded"

4. Some operating systems impose limits on the length of lines within a text file. To circumvent this limitation it is possible to use continuation lines. These lines end with a backslash and newline. In a quoted argument, continuation lines will be appended without stripping any whitespace on the next line. For non-quoted arguments, all whitespace on the next line will be stripped.

Example:

```
"This is a continuation \  
line"  
-> "This is a continuation line"  
  
control(file1(mode,type),\  
        file2(type))  
->  
control(file1(mode,type),file2(type))
```

5. It is possible to nest command line files up to 25 levels.

Example:

Suppose the file `mycmds` contains the following line:

```
-err  
test.c
```

The command line can now be:

```
ctrl -f mycmds
```

-FPU

Option:



Select the Project | C Compiler Options | Project Options... menu item. Enable the Use hardware single precision floating point instructions check box in the Misc tab.



This option is only available (and relevant) when you enable the FPU present (on user defined CPU) check box on the CPU tab in the Project | Processor Options... menu item.



-FPU

Description:

The **-FPU** option allows the generation of single precision floating point instructions in the generated assembly file. When you select this option, the macro `_FPU` is defined in the C source file. For a more detailed description about the floating point arithmetic see Section 7.5, *Floating Point Arithmetic* in Chapter 7, *Run-time Environment*.

Example:

To allow the use of floating point unit (FPU) instructions in the generated assembly file, enter:

```
ctri -FPU test.c
```

-g

Option:



Select the Project | C Compiler Options | Project Options... menu item. Enable the Generate symbolic debug information check box in the Debug tab. Optionally enable the Include debug information for non referenced types and Disable lifetime info for all types check boxes.



-g[f|l|n]...

Description:

Add directives to the output files, incorporating symbolic information to facilitate high level debugging.

With **-gn** you disable all debug, including type checking.

With **-gl** you disable lifetime information for all types.

If you use **-gf**, high level language type information is also emitted for types which are not referenced by variables. Therefore, this sub-option is not recommended.

Examples:

To add symbolic debug information to the output files, enter:

```
ctri -g test.c
```

To add symbolic debug information to the output files but disable lifetime information for all types, enter:

```
ctri -gl test.c
```

To disable all symbolic debug information including type checking, enter:

```
ctri -gn test.c
```

-H

Option:



Select the Project | C Compiler Options | Project Options... menu item. Enter a filename in the First #include this file field in the Preprocessing tab.



-H*file*

Arguments:

The name of an include file.

Description:

Include *file* before compiling the C-source. This is the same as specifying #include "*file*" at the first line of your C-source.

Example:

```
ctri -Hstdio.h test.c
```



-I



Option:



Select the Project | C Compiler Options | Project Options... menu item. Enter one or more directory paths to the Include search path field in the Preprocessing tab.



-I*directory*

Arguments:

The name of the directory to search for include file(s).

Description:

Change the algorithm for searching `#include` files whose names do not have an absolute pathname to look in *directory*. Thus, `#include` files whose names are enclosed in `""` are searched for first in the directory of the file containing the `#include` line, then in directories named in **-I** options in left-to-right order. If the include file is still not found, the compiler searches in a directory specified with the environment variable CTRIINC. CTRIINC may contain more than one directory. Finally, the directory `../include` relative to the directory where the compiler binary is located is searched. This is the standard include directory supplied with the compiler package.

For `#include` files whose names are in `<>`, the directory of the file containing the `#include` line is not searched. However, the directories named in **-I** options (and the one in CTRIINC and the relative path) are still searched.

Example:

```
ctri -I/proj/include test.c
```



Section *Include Files*.

-indirect

Option:



Select the Project | C Compiler Options | Project Options... menu item. Enable the Call functions indirect check box in the Misc tab.



-indirect

Description:

With the **-indirect** option you can globally enable code generation for indirect function calling.

Example:

```
ctri -indirect test.c
```



See Section 3.9.7, *Far Function Storage Qualifier*.

-K

Option:



Select the Project | C Compiler Options | Project Options... menu item. Disable the Type checking for old (K&R) function calls check box in the Misc tab.



-K

Description:

Do not perform type-checking for old-style (K&R) function calls.

When a function is called without a visible prototype, the compiler will synthesize a prototype from the types of the function arguments. This prototype is then used for type checking of further calls to the same function. The **-K** option disables this type checking.

Example:

Consider the following two calls to a function without a visible prototype:

```
unknown(1,2);
unknown(3);
```

The compiler will issue an error message for the second call, because it is not compatible with the prototype synthesized for the first call.

-misrac

Option:



Select the Project | MISRA C Compiler Options | Project Options... menu item. Select one of the MISRA C options. If you select Custom MISRA C configuration you can enable or disable specific rules in the numbered tabs.



-misrac*n,n,...*

Arguments:

The MISRA C rules to be checked.

Description:

With this option, the MISRA C rules to be checked can be specified. Refer to Appendix B *MISRA C* for a list of supported and unsupported MISRA C rules.

Example:

```
ctri -misrac9 test.c
```

Will generate an error in case 'test.c' contains nested comments.

-N

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select a `_near` allocation radio button in the Allocation tab. Optionally enter a threshold value.



-N[*threshold*]

Arguments:

Optionally the threshold size of an object (in bytes).

Default:

-N8

Description:

Specify the threshold for `_near`/`_far` allocation. When you do not specify either `_near` or `_far` in a declaration, the compiler uses a default based on the size of the object. Objects smaller or equal to the threshold will be allocated `_near`. Larger objects, arrays and strings will be allocated `_far`.

The default threshold is eight bytes.

When the **-N** option is specified without a threshold value, all objects will be allocated `_near`., including arrays and string constants.

Example:

To specify a threshold of 12 bytes, enter:

```
ctri -N12 test.c
```

-n

Option:

-n

Description:

Do not create output files; instead, the output is sent to stdout.

Example:

```
ctri -n test.c
```

-O

Option:



Select the Project | C Compiler Options | Project Options... menu item. You can control optimizations in the Optimization tab.



-O*flags*

Pragma:

optimize *flags*

Arguments:

One or more optimization flags.

Default:

-O1

Description:

Control optimization. If you do not use this option, the default optimization of **ctri** is **-O1**, which is an optimization level to let **ctri** generate the smallest code.

Flags which are controlled by a letter, can be switched on with the lower case letter and switched off with the uppercase letter. These options are described together.

All optimization flags can also be given in the source file after a `#pragma optimize`.

An overview of the flags is given below.

- a** – relax alias checking (needs **-Oc**)
 - c** – common subexpression elimination
 - e** – expression propagation (needs **-Oc**)
 - f** – code flow, order rearranging
 - i** – move invariant code outside loop (needs **-Oc**)
 - l** – fast loops (increases code size)
 - o** – software pipelining (needs **-Ocfv**, increases code size)
 - p** – data flow, constant/copy propagation
 - s** – optimize for speed
 - t** – tail call conversion
 - u** – loop unrolling
 - v** – subscript strength reduction
 - w** – global variable writeback caching
 - y** – peephole optimization
 - z** – pipeline scheduler
-
- 0** – same as **-OACEFILOPSTUVWYZ** (no optim)
 - 1** – same as **-OacefiLOpStUvwyZ** (default, size)
 - 2** – same as **-OacefilopstUvwyZ** (speed)

Example:

```
ctri -OAcFiLPv test.c
```



Pragma optimize in section *Pragmas*.

-Onumber

Option:



Select the Project | C Compiler Options | Project Options... menu item. Choose an Optimization level in the Optimization tab.



-Onumber

Arguments:

A number in the range 0 – 2.

Default:

-O1

Description:

Control optimization. You can specify a single number in the range 0 – 2, to enable or disable optimization. The options are a combination of the other optimization flags:

- O0** – same as **-OACEFILOPSTUVWYZ**
Switchable optimizations switched off
- O1** – same as **-OacefiLOpStUvwyz**
Default. Set optimization flages to let **ctri** generate the smallest code.
- O2** – same as **-OacefilopstUvwyz**
Set optimization flags to let **ctri** generate the fastest code.



The flags 0 to 2 cannot be concatenated with other flags. For example, **-Oa2c** is not allowed, **-OacF** is allowed.

Example:

To optimize for code size, enter:

```
ctri -O1 test.c
```

-Oa / -OA

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Relax alias checking check box.



-Oa / -OA

Pragma:

optimize a / optimize A

Default:

-OA

Description:

With **-Oa** you relax alias checking. If you specify this option, **ctri** will not erase remembered register contents of user variables if a write operation is done via an indirect (calculated) address to a variable with a different type.



It is quite safe to enable this optimization, unless you are assigning the address of a variable to a pointer of a different type.

With **-OA** you specify strict alias checking. If you specify this option, the compiler erases all register contents of user variables when a write operation is done via an indirect (calculated) address.



Pragma optimize in section *Pragmas*.

-Oc / -OC

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Common subexpression elimination (CSE) check box.



-Oc / -OC

Pragma:

optimize c / optimize C

Default:

-Oc

Description:

With **-Oc** you enable CSE (common subexpression elimination). With this option specified, the compiler tries to detect common subexpressions within the C code. The common expressions are evaluated only once, and their result is temporarily held in registers.

Note that the **-Oc** option must be on to enable moving invariant code outside a loop (**-Oi**).

With **-OC** you disable CSE (common subexpression elimination). With this option specified, the compiler will not try to search for common expressions. Also expression propagation and moving invariant code outside a loop will be disabled.

Example:

```
/*
 * Compile with -OC -O0,
 * Compile with -Oc -O0, common subexpressions are found
 *   and temporarily saved.
 */

char x, y, a, b, c, d;

void
main( void )
{
    x = (a * b) - (c * d);

    y = (a * b) + (c * d); /*(a*b) and (c*d) are common */
}
```



Pragma optimize in section *Pragmas*.

-Oe / -OE

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Expression propagation check box.



-Oe / -OE

Pragma:

optimize e / optimize E

Default:

-Oe

Description:

With **-Oe** you enable expression propagation. With this option, the compiler tries to find assignments of expressions to a variable, a subsequent assignment of the variable to another variable can be replaced by the expression itself. Note that the option **-Oc** must be on to use this option.

With **-OE** you disable expression propagation.

Example:

```
/*
 * Compile with -OE -Oc -O0, normal cse is done
 * Compile with -Oe -Oc -O0, 'i+j' is propagated.
 */

unsigned i, j;

int
main( void )
{
    static int a;
    a = i + j;
    return (a);
}
```



-Oc

Pragma optimize in section *Pragmas*.

-Of / -OF

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Control flow optimization and code reordering check box.



-Of / -OF

Pragma:

optimize f / optimize F

Default:

-Of

Description:

With **-Of** you enable control flow optimizations and code order rearranging on the intermediate code representation, such as jump chaining and conditional jump reversal.

With **-OF** you disable control flow optimizations.

Examples:

The following example shows a control optimization:

```
/*
 * Compile with -OF -O0
 * Compile with -Of -O0, compiler finds first time 'i'
 * is always < 10, the unconditional jump is removed.
 */
int i;

void
main( void )
{
    for( i=0; i<10; i++ )
    {
        do_something();
    }
}
```

The following example shows a conditional jump reversal:

```
/*
 * Compile with -Of -O0, code as written sequential
 * Compile with -Of -O0, code is rearranged
 *
 * Code rearranging enables other optimizations to
 * optimize better, e.g. CSE
 */

int i;
extern void dummy( void );

void main ()
{
    do
    {
        if ( i )
        {
            i--;
        }
        else
        {
            i++;
            break;
        }
        dummy();
    } while ( i );
}
```



Pragma optimize in section *Pragmas*.

-Oi / -OI

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Move invariant code outside loop check box.



-Oi / -OI

Pragma:

optimize i / optimize I

Default:

-Oi

Description:

With **-Oi** you move invariant code outside a loop. Note that the option **-Oc** must be on to use this option.

With **-OI** you disable moving invariant code outside a loop.

Example:

```
/*
 * Compile with -OI -Oc -O0, normal cse is done
 * Compile with -Oi -Oc -O0, invariant code is found in
 *   the loop, code is moved outside the loop.
 */
void
main( void )
{
    char x, y, a, b;
    int i;

    for( i=0; i<20; i++ )
    {
        x = a + b;
        y = a + b;
    }
}
```



-Oc

Pragma optimize in section *Pragmas*.

-OI / -OL

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Fast loops (more code size) check box.



-OI / -OL

Pragma:

optimize 1 / optimize L

Default:

-OL

Description:

With **-OI** you enable fast loops. Duplicate the loop condition. Evaluate the loop condition one time outside the loop, just before entering the loop, and at the bottom of the loop. This saves one unconditional jump and gives less code inside a loop.

With **-OL** you disable fast loops.

Example:

```
/*
 * Compile with -OL -O0
 * Compile with -OI -O0, compiler duplicates the loop
 * condition, the unconditional jump is removed.
 */
int i;

void
main( void )
{
    for( ; i<10; i++ )
    {
        do_something();
    }
}
```



Pragma optimize in section *Pragmas*.

-Oo / -OO

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Software pipelining (increases code size) check box.



-Oo / -OO

Pragma:

optimize o / optimize O

Default:

-OO

Description:

With **-Oo** you enable software pipelining. Software Pipelining is a technique which increases parallelism in loops by executing iterations in an overlapped fashion. This includes the use of SIMD (single instruction multiple data) instructions. The optimization requires **-Ocfv**. Loops may execute up to four times faster with this optimization at the expense of an increased code size.

A loop must meet the following requirements:

- It should merely operate on `_sfract` data types including at least one array variable.
- The loop trip count must be a compile time constant.
- The loop should not contain conditional code, references to volatile objects, inline assembly or function calls. Calls to inline or intrinsic functions are permitted but may degrade the optimization.

When the loop updates an array it is necessary in most cases to tell the compiler that the array cannot overlap with other arrays used in the same loop. This can be achieved by declaring the arrays as `restrict` qualified pointers.

With **-OO** you disable software pipelining.

Example:

```
/*
 * With software pipelining enabled two iterations
 * will be done in parallel using SIMD instructions.
 */

_sat _sfract  acc;
void vquant(_sat _sfract *x, _sat _sfract *k)
{
    int i;

    acc = 0;
    for (i = 0; i < 64; ++i)
    {
        acc += (k[i] - x[i]) * (k[i] - x[i]);
    }
}
```



Pragma optimize in section *Pragmas*.

-Op / -OP

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Constant and copy propagation check box.



-Op / -OP

Pragma:

optimize p / optimize P

Default:

-Op

Description:

With **-Op** you enable constant and copy propagation. With this option, the compiler tries to find assignments of constant values to a variable, a subsequent assignment of the variable to another variable can be replaced by the constant value.

With **-OP** you disable constant and copy propagation.

Example:

```
/*
 * Compile with -OP -O0, 'i' is actually assigned to 'j'
 * Compile with -Op -O0, 15 is assigned to 'j', 'i' was
 * propagated
 */

int i;
int j;

void
main( void )
{
    i = 10;
    j = i + 5;
}
```



Pragma optimize in section *Pragmas*.

-Os / -OS

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Optimize for speed instead of size check box.



-Os / -OS

Pragma:

optimize s / optimize S

Default:

-Os

Description:

With **-Os** you tell the compiler to generate faster code. Whenever possible instructions are used that use less instruction cycles.

With **-OS** you disable optimization for speed, favor code size over execution speed.



Pragma optimize in section *Pragmas*.

-Ot / -OT

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Tail call conversion check box.



-Ot / -OT

Pragma:

optimize t / optimize T

Default:

-Ot

Description:

With **-Ot** you tell the compiler to replace a CALL directly followed by a RET with a J instruction.

With **-OT** you disable this optimization. This may be the case for debugging.



Pragma optimize in section *Pragmas*.

-Ou / -OU

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Loop unrolling check box.



-Ou / -OU

Pragma:

optimize u / optimize U

Default:

-OU

Description:

With **-Ou** you enable loop unrolling. With this option specified, the compiler tries to eliminate short loops by duplicating a loop body 2, 4 or 8 times. This reduces the number of branches and creates a longer linear code part.

With **-OU** you disable loop unrolling.

Example:

```
/*
 * Compile with -OU, normal loop handling
 * Compile with -Ou, loop is eliminated, body is duplicated
 */
int i, j;

void
main( void )
{
    for( i=0; i<2; i++ )      /* short loop */
    {
        j = 2 * i;
    }
}
```



Pragma optimize in section *Pragmas*.

-Ov / -OV

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Subscript strength reduction check box.



-Ov / -OV

Pragma:

optimize v / optimize V

Default:

-Ov

Description:

With **-Ov** you enable subscript strength reduction. With this option specified, the compiler tries to reduce expressions involving an index variable in strength.

With **-OV** you disable subscript strength reduction.

Example:

```
/*
 * Compile with -OV -O0, disable subscript strength reduction
 * Compile with -Ov -O0, begin and end address of 'a' are
 * determined before the loop and temporarily put in registers
 * instead of determining the address each time inside the loop
 */
int a[4];

void main( void )
{
    int i;
    for( i=0; i<4; i++ )
    {
        a[i] = 0;
    }
}
```



Pragma optimize in section *Pragmas*.

-Ow / -OW

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Global variable writeback caching check box in the ... Optimization tab.



-Ow / -OW

Pragma:

optimize w / optimize W

Default:

-Ow

Description:

With **-Ow** the compiler will cache multiple writes to a global variable in a register in some situations.

With **-OW** all updates of a global variable are directly written to memory.

Examples:

```
/*
 * Compile with -OW -O0, 'sum' is updated twice
 * Compile with -Ow -O0, 'sum' is updated only once
 */
int sum;

void
add( int a, int b )
{
    sum += a;
    sum += b;
}
```



Pragma optimize in section *Pragmas*.

-Oy / -OY

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Peephole optimizer (remove redundant code) check box in the ... Optimization tab.



-Oy / -OY

Pragma:

optimize y / optimize Y

Default:

-Oy

Description:

With **-Oy** you enable peephole optimization. Remove redundant code. The peephole optimizer searches for redundant instructions or for instruction sequences which can be combined to minimize the number of instructions.

With **-OY** you disable peephole optimization.



Pragma optimize in section *Pragmas*.

-Oz / -OZ

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select the Advanced optimization level in the Optimization tab. Enable or disable the Instruction pipeline scheduler check box in the ... Optimization tab.



-Oz / -OZ

Pragma:

optimize z / optimize Z

Default:

-Oz

Description:

With **-Oz** you enable instruction scheduling. The instruction scheduler will try to rearrange instructions in order to make better use of the parallel execution capabilities of the TriCore CPU. As a result, the program will run faster. A disadvantage of this optimization is that the assembly code generated by the C compiler is more difficult to read.

With **-OZ** you disable instruction scheduling.



Pragma optimize in section *Pragmas*.

-o

Option:

-o *file*

Arguments:

An output filename. The filename may not start immediately after the option. There must be a tab or space in between.

Default:

Module name with `.src` suffix.

Description:

Use *file* as output filename, instead of the module name with `.src` suffix. Special care must be taken when using this option, the first **-o** option found acts on the first file to compile, the second **-o** option acts on the second file to compile, etc.

Example:

When specified:

```
ctri file1.c file2.c -o file3.src -o file2.src
```

two files will be created, **file3.src** for the compiled file **file1.c** and **file2.src** for the compiled file **file2.c**.

-R

Option:



Select the Project | C Compiler Options | Project Options... menu item. Add the option to the Additional options field in the Misc tab.



-R[*name*]

Pragma:

section

Arguments:

Optionally, a section name.

Description:

The compiler defaults to a section naming convention, using a memory type abbreviation and the module name: *code.name* for code sections, *bss.name* for cleared direct addressable data, etc. In case a module must be loaded at a fixed address or a data section needs a special place in memory, you can use the **-R** option to generate a different section name. This way the order **lctri** allocates these sections can be specified in the locator description file.

Example:

To generate the section name *section_type.NEW* instead of the default section name *section_type.mod_name*, enter:

```
ctri -RNEW test.c
```

To generate the section name *section_type* instead of the default section name *section_type.mod_name*, enter:

```
ctri -R test.c
```

-S

Option:



Select the Project | C Compiler Options | Project Options... menu item. Enable the Merge C source code with assembly in output file (.src) check box in the Output tab.



-s

Pragma:

source

Description:

Merge C source code with generated assembly code in output file.

Example:

```
ctri -s test.c

;          test.c:
; 1      |int i;
; 2      |
; 3      |int
; 4      |main( void )
; 5      |{

        .global  main
        .extern  _START
```



Pragmas source and nosource in section *Pragmas*.

-TC2

Option:



Select the **Project | Processor Options...** menu item. Select the CPU tab. Select TC2 in the CPU type box.



-TC2

Description:

The **-TC2** option allows the generation of TriCore2 instructions in the generated assembly file. When you select this option, the macro `_TC2` is defined in the C source file.

Example:

To allow the use of TriCore2 instructions in the generated assembly file, enter:

```
ctri -TC2 test.c
```

-U

Option:



Select the Project | C Compiler Options | Project Options... menu item. Undefine one or more of the predefined symbols `_TASKING` or `_CTRI` by disabling the corresponding check box in the Preprocessing tab.



-Uname

Arguments:

The name macro you want to undefine.

Description:

Remove any initial definition of identifier *name* as in `#undef`, unless it is a predefined ANSI standard macro. ANSI specifies the following predefined symbols to exist, which cannot be removed:

<code>__FILE__</code>	"current source filename"
<code>__LINE__</code>	current source line number (int type)
<code>__TIME__</code>	"hh:mm:ss"
<code>__DATE__</code>	"Mmm dd yyyy"
<code>__STDC__</code>	level of ANSI standard. This macro is set to 1 when the option to disable language extensions (-A) is effective. Whenever language extensions are excepted, <code>__STDC__</code> is set to 0 (zero).

When **ctri** is invoked, also the following predefined symbols exist:

<code>_CTRI</code>	predefined symbol to identify the compiler. This symbol can be used to flag parts of the source which must be recognized by the ctri compiler only. It expands to the version number of the compiler.
<code>_TASKING</code>	identifies the compiler. This symbol can be used to flag parts of the source which must be recognized by TASKINGs ctri only. It expands to 1.

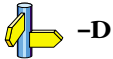
`_SINGLE_FP` This symbol is defined when the **-F** option is used to map type `double` onto type `float`.

`_DOUBLE_FP` This symbol is defined when the **-F** option is not used.

These symbols can be turned off with the **-U** option.

Example:

```
ctri -U_TASKING test.c
```



-u

Option:



Select the Project | C Compiler Options | Project Options... menu item. Enable the Treat 'char' variables as unsigned check box in the Misc tab.



-u

Description:

Treat 'character' type variables as 'unsigned character' variables. By default char is the same as specifying signed char. With **-u** char is the same as unsigned char.

Example:

With the following command char is treated as unsigned char:

```
ctri -u test.c
```

-V

Option:

-V

Description:

Display version information.

Example:

```
ctri -V
```

```
TASKING TriCore C compiler      vx.yrz Build nnn  
Copyright 1996-year Altium BV   Serial# 00000000
```

-WAE

Option:

-WAE

Description:

Treat warning messages as errors. This also affects the return value of the application when only warnings occur. A build process will now stop when warnings occur.

Example:

```
ctri -WAE test.c
```

-W

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select one of the Diagnostics options in the Output tab and optionally fill in specific message numbers to suppress.



-w*num*
-wstrict

Arguments:

Optionally the warning number to suppress.

Description:

-w suppress all warning messages. **-w***num* only suppresses the given warning. **-wstrict** suppresses all "strict" warning messages (183, 196, 303).

Example:

To suppress warning 135, enter:

```
ctri file1.c -w135
```

-Z

Option:



Select the Project | C Compiler Options | Project Options... menu item. Select a `_near` allocation radio button in the Allocation tab. Optionally enter a threshold value.



-Z[*threshold*]

Arguments:

Optionally the threshold size of an object (in bytes).

Default:

-Z0

Description:

Specify the threshold for `_a0` allocation. When you do not specify any memory specifier such as `_near` or `_far` in a declaration, the compiler uses a default based on the size of the object

First, the size of the object is checked against the **-N** threshold, according to the description of the **-N** option. If the size is larger than the **-N** threshold, but lower or equal to the **-Z** threshold, the object is allocated in `_a0` memory. Larger objects, arrays and strings will be allocated `_far`.

The default **-Z** threshold is zero, which means that the compiler will never use `_a0` memory unless you specify the **-Z** option. When you use the **-Z** option without a threshold value, all objects not allocated `_near`, including arrays and string constants, will be allocated in `_a0` memory.

Allocation in `_a0` memory means that the object is addressed indirectly, using A0 as the base pointer. The total amount of memory that can be addressed this way is 64 Kbytes.

Example:

To specify a threshold of 12 bytes, enter:

```
ctri -Z12 test.c
```

-Z

Option:



Select the Project | C Compiler Options | Project Options... menu item. Add the option to the Additional options field in the Misc tab.



-zpragma

Arguments:

A pragma. The compiler recognizes all pragmas that are mentioned in section 4.4 *Pragmas* as well as pragmas that can be used to solve CPU functional problems (see Appendix C, *CPU Functional Problems*).

Description:

With this option you can give a pragma on the command line. This is the same as specifying '#pragma *pragma*' in the C source. Dashes ('-') on the command line in the pragma are converted to spaces. A dash prefixed by another dash or space is never translated, so it is still possible to specify a dash for negative numbers as pragma argument.

If you use quotes inside a quoted argument, use the following syntax:

- a. If you use a single quote, surround the argument with double quotes.
If you use a double quote, surround the argument with single quotes.

Example:

"Single quote ' embedded"

'Double quote " embedded'

- b. If you use both types of quotes, split the argument in such a way that each embedded quote is surrounded by the opposite type of quote.

'Double quote " and single quote ''' embedded"

Example:

The following option

-zoptimize-p

is equivalent with:

#pragma optimize p



To include pragmas to enable bypasses for CPU functional problems, you can also use EDE:

Select the Project | Processor Options... menu and choose the Bypasses TC1 v1.2 tab or the Bypasses TC1 v1.3 tab. Then select the bypasses you want to enable.



Section 4.4, *Pragmas*,
Appendix C, *CPU Functional Problems*.

4.3 INCLUDE FILES

You may specify include files in two ways: enclosed in `<>` or enclosed in `""`. When an `#include` directive is seen, **ctri** used the following algorithm trying to open the include file:

1. If the filename is enclosed in `""`, and it is not an absolute pathname (does not begin with a `'\'` for PC, or a `'/'` for UNIX), the include file is searched for in the directory of the file containing the `#include` line. For example, in:

PC:

```
ctri ../../source/test.c
```

UNIX:

```
ctri ../../source/test.c
```

ctri first searches in the directory `../../source` (`../../source` for UNIX) for include files.

If you compile a source file in the directory where the file is located (**ctri** **test.c**), the compiler searches for include files in the current directory.



This first step is not done for include files enclosed in `<>`.

2. Use the directories specified with the **-I** options, in a left-to-right order. For example:

PC:

```
ctri -I../../include demo.c
```

UNIX:

```
ctri -I../../include demo.c
```

3. Check if the environment variable **CTRIINC** exists. If it does exist, use the contents as a directory specifier for include files. You can specify more than one directory in the environment variable **CTRIINC** by using a separator character. Instead of using **-I** as in the example above, you can specify the same directory using **CTRIINC**:

PC:

```
set CTRIINC=..\..\include
ctri demo.c
```

UNIX:

if using the Bourne shell (sh)

```
CTRIINC=../../include
export CTRIINC
ctri demo.c
```

or if using the C-shell (csh)

```
setenv CTRIINC ../../include
ctri demo.c
```

4. When an include file is not found with the rules mentioned above, the compiler tries the subdirectory `include`, one directory higher than the directory containing the **ctri** binary. For example:

PC:

ctri.exe is installed in the directory `C:\CTRI\BIN`
The directory searched for the include file is `C:\CTRI\INCLUDE`

UNIX:

ctri is installed in the directory `/usr/local/ctri/bin`
The directory searched for the include file is
`/usr/local/ctri/include`

The compiler determines run-time which directory the binary is executed from to find this `include` directory.

A directory name specified with the **-I** option or in `CTRIINC` may or may not be terminated with a directory separator, because **ctri** inserts this separator, if omitted.

When you specify more than one directory to the environment variable `CTRIINC`, you have to use one of the following separator characters:

PC:

`;` , *space*

e.g. `set CTRIINC=..\..\include;\proj\include`

UNIX:

: ; , *space*

e.g. **setenv CTRIINC ../../include:/proj/include**

4.4 PRAGMAS

According to ANSI (3.8.6) a preprocessing directive of the form:

```
#pragma pragma-token-list new-line
```

causes the compiler to behave in an implementation-defined manner. The compiler recognizes all pragmas that are mentioned below as well as pragmas that can be used to solve CPU functional problems (see Appendix C, *CPU Functional Problems*). Other pragmas will be ignored. Pragmas give directions to the code generator of the compiler. Besides the pragmas there are two other possibilities to steer the code generation process: command line options and keywords (e.g., `_near` type variables) in the C application itself. The compiler acknowledges these three groups using the following rules:

Command line options can be overruled by keywords and pragmas. Keywords can be overruled by pragmas. Hence, pragmas have the highest priority.

This approach makes it possible to set a default optimization level for a source module, which can be overridden temporarily within the source by a pragma.

The C compiler **ctri** supports the following pragmas:

align

Align objects of four bytes or larger to a word boundary. By default, the minimum alignment dictated by the hardware is used. Depending on the cache configuration, using a larger alignment may be faster.

This pragma optionally accepts one of the following arguments:

on	enable this pragma (default)
off	disable this pragma
<i>n</i>	align object of size four or larger to <i>n</i> (<i>n</i> must be a power of two)
restore	restore the previous setting.

asm

Insert the following (non preprocessor lines) as assembly language source code into the output file. The inserted lines are not checked for their syntax. The code buffer of the peephole optimizer is flushed. Thus the compiler will stop optimizations like peephole pattern replacement and resumes these optimizations after the **endasm** pragma as if it starts at the beginning of a function.



See section 3.15, *Inline Assembly*, for a description of the optional C variable interface.

For advanced assembly in–lining, intrinsic functions can be used. The defined set of intrinsic functions cover most of the specific TriCore features which could otherwise not be accessed by the C language.



For more information on intrinsic functions see section 3.16, *Intrinsic Functions*.

asm_noflush

Same as pragma **asm**, except that the optimizer does not flush its information at this point, so that copy and constant propagation may be performed across the inline assembly fragment.

clear

Perform 'clearing' of non–initialized static/public variables.

noclear

No 'clearing' of non–initialized static/public variables.

endasm

Switch back to the C language.

intenum

Force all enumeration types to type `int`. By default, the compiler uses `char` or `short` when that is sufficient to represent all enum values.

This pragma optionally accepts one of the following arguments:

on	enable this pragma
off	disable this pragma (default)
restore	restore the previous setting.

listinc

Expand include files in generated list file.

nolistinc

Default. Do not expand include files in list file.

optimize flags

Controls the amount of optimization. The remainder of the source line is scanned for option characters, which are processed like the flags of the **-O** command line option. Please refer to the **-O** option for the list of available flags.

optimize restore

End a region that was optimized with a **#pragma optimize**. The pragma **optimize restore** restores the situation as it was before the corresponding pragma **optimize**. **#pragma optimize/optimize restore** pairs can be nested.

Example:

```
#pragma optimize 0
    /* disable all optimizations */
    . . .
#pragma optimize 2
    /* enable all optimizations */
switch(...)
{
    . . .
}
#pragma optimize restore
    /* back to all optimizations disabled */
#pragma optimize restore
    /* back to default optimizations */
```

pack 2

Qualify a structure or union type for halfword packing. The structure or union to be packed cannot have members that require an alignment larger than 2-byte.

Example:

```
#pragma pack 2
typedef struct {
    unsigned char  uc1;
    unsigned char  uc2;
    unsigned short us1;
    unsigned short us2;
    unsigned short us3;
} packed_struct;
#pragma pack 0
```

The **#pragma pack** directives can be nested.



See also section 3.6.3, *Halfword Packed Unions and Structures* in Chapter 3, *Language Implementation*.

pack 0

Turn off halfword packing for structures and unions.

When you place a **#pragma 0** before a structure or union, its alignment will not be changed:

```
#pragma pack 0
packed_struct pstruct;
```

section [sect_type=]"name"

Change the default section names for new objects. The default name used by the compiler consists of a prefix denoting the section type, and a suffix derived from the module name. For example, the default name for the code section of the module `hello.c` is `code.hello`. After a **#pragma section**, the current state will be the default situation plus all modifications from the last **#pragma section**. All modifications from previous **#pragma section** statements will be lost.

With **#pragma section**, you can change the default section name suffix or specify an alternative name for specific sections. To change the default suffix, specify the new suffix as a string after the `pragma`:

```
#pragma section "my_suffix"
```



You can also change the default section name suffix from the command line with the **-R** option.

To change a specific section name, you have to specify the section type, an equal sign, followed by the new section name. For example:

```
#pragma section code="my_code" data="my_data"
```

The following section types are available:

Type	Description
code	program code
data	initialized _near data
fardata	initialized _far data
bss	uninitialized _near data (cleared)
farbss	uninitialized _far data (cleared)
rom	constant _near data
farrom	constant _far data
noclear	uninitialized _near data
farnoclear	uninitialized _far data

Table 4-4: Section types

```
#pragma section code_init
```

With the pragma above, the code section is copied from ROM to RAM when the program is started.

```
#pragma section data_overlay
```

The pragma above allows the `noclear` and `farnoclear` section to be overlayed with other sections with the same name. Because of the default naming scheme, you must force the existence of identical section names with **#pragma section** [*sect_type=*]*"name"*. The use of the `noclear` sections must be forced with **#pragma noclear**.

The following pragmas allow the `noclear` overlay and the `farnoclear` overlay sections to be overlayed with similar named sections in other modules:

```
#pragma noclear
#pragma section data_overlay "overlay"
```

To restore the default section naming scheme, use:

```
#pragma section
```


source

Same as **-s** option. Enable mixing C source with assembly code.

nosource

Default. Disable generation of C source within assembly code.

stack

Use the alternative "stack" calling convention for the function or function pointer immediately following this pragma. This pragma is equivalent with the function qualifier **_stackparm**.

switch auto

Default. The compiler decides what code to generate on a switch statement. In general, the compiler chooses the most efficient method in terms of code density and execution speed. See also the paragraph *Switch Statement*.

switch linear

Force the compiler to generate linear jump code for switch statements. See also the paragraph *Switch Statement*.

switch jumptab

Force the compiler to generate jump tables for switch statements. See also the paragraph *Switch Statement*.

switch lookup

Force the compiler to generate lookup tables for switch statements. See also the paragraph *Switch Statement*.

switch restore

The compiler restores the previous switch method. See also the paragraph *Switch Statement*. Example:

```
#pragma switch linear
    /* linear jump code */
    . . .
#pragma switch lookup
    /* lookup tables */
switch(...)
{
    . . .
}
#pragma switch restore
    /* back to switch linear */
#pragma switch restore
    /* back to default switch auto */
```

TC112_DEFECTS

Enables all C compiler bypasses for the TC112 CPU functional problems.

TC113_DEFECTS

Enables all C compiler bypasses for the TC113 CPU functional problems.

Pragmas are also available to enable individual bypasses for CPU functional problems. These pragmas have the names of the functional problems as defined by Infineon. Refer to Appendix C, *CPU Functional Problems* for a complete overview of these pragmas.

4.5 COMPILER LIMITS

The ANSI C standard [1-2.2.4] defines a number of translation limits, which a C compiler must support to conform to the standard. The standard states that a compiler implementation should be able to translate and execute a program that contains at least one instance of every one of the limits listed below. **ctri**'s actual limits are given within parentheses.

Most of the actual compiler limits are determined by the amount of free memory in the host system. In this case a 'D' (Dynamic) is given between parentheses. Some limits are determined by the size of the internal compiler parser stack. These limits are marked with a 'P'. Although the size of this stack is 200, the actual limit can be lower and depends on the structure of the translated program.

- 15 nesting levels of compound statements, iteration control structures and selection control structures (P > 15)
- 8 nesting levels of conditional inclusion (50)
- 12 pointer, array, and function declarators (in any combinations) modifying an arithmetic, a structure, a union, or an incomplete type in a declaration (15)
- 31 nesting levels of parenthesized declarators within a full declarator (P > 31)
- 32 nesting levels of parenthesized expressions within a full expression (P > 32)
- 31 significant characters in an external identifier (full ANSI-C mode), 1500 significant characters in an external identifier (non ANSI-C mode)
- 511 external identifiers in one translation unit (D)
- 127 identifiers with block scope declared in one block (D)
- 1024 macro identifiers simultaneously defined in one translation unit (D)
- 31 parameters in one function declaration (D)
- 31 arguments in one function call (D)
- 31 parameters in one macro definition (D)
- 31 arguments in one macro call (D)
- 509 characters in a logical source line (1500)
- 509 characters in a character string literal or wide string literal (after concatenation) (1500)
- 8 nesting levels for **#included** files (50)
- 257 case labels for a switch statement, excluding those for any nested switch statements (D)

- 127 members in a single structure or union (D)
- 127 enumeration constants in a single enumeration (D)
- 15 levels of nested structure or union definitions in a single struct-declaration-list (D)



CHAPTER

5

COMPILER DIAGNOSTICS



5

CHAPTER

5.1 INTRODUCTION

ctri has three classes of messages: user errors, warnings and internal compiler errors.

Some user error messages carry extra information, which is displayed by the compiler after the normal message. The messages with extra information are marked with 'I' in the list below. They never appear without a previous error message and error number. The number of the information message is not important, and therefore, this number is not displayed. A user error can also be fatal (marked as 'F' in the list below), which means that the compiler aborts compilation immediately after displaying the error message and may generate a 'not complete' output file.

The error numbers and warning numbers are divided in two groups. The frontend part of the compiler uses numbers in the range 0 to 499, whereas the backend (code generator) part of the compiler uses numbers in the range 500 and higher. Note that most error messages and warning messages are produced by the frontend.

If you program a non fatal error, **ctri** displays the C source line that contains the error, the error number and the error message on the screen. If the error is generated by the code generator, the C source line displayed always is the last line of the current C function, because code generation is started when the end of the function is reached by the frontend. However, in this case, **ctri** displays the line number causing the error before the error message. **ctri** always generates the error number in the assembly output file, exactly matching the place where the error occurred.

So, when a compilation is not successful, the generated output file is not accepted by the assembler, thus preventing a corrupt application to be made (see also the **-e** option).

Warning messages do not result into an erroneous assembly output file. They are meant to draw your attention to assumptions of the compiler, for a situation which may not be correct. Warning messages can be controlled with the **-w[num]** option.

The last class of messages are the internal compiler errors. The following format is used:

S number: internal error - please report

These errors are caused by failed internal consistency checks and should never occur. However, if such a 'SYSTEM' error appears, please report the occurrence to TASKING, using a Problem Report form. Please include a diskette or tape, containing a small C program causing the error.

5.2 RETURN VALUES

ctri returns an exit status to the operating system environment for testing.

For example,

in a BATCH-file you can examine the exit status of the program executed with ERRORLEVEL:

```
ctri -s %1.c
IF ERRORLEVEL 1 GOTO STOP_BATCH
```

In a bourne shell script, the exit status can be found in the **\$?** variable, for example:

```
ctri $*
case $? in
0)      echo ok ;;
1|2|3)  echo error ;;
esac
```

The exit status of **ctri** is one of the numbers of the following list:

- 0 Compilation successful, no errors
- 1 There were user errors, but terminated normally
- 2 A fatal error, or System error occurred, premature ending
- 3 Stopped due to user abort

5.3 ERRORS AND WARNINGS

Errors start with an error type, followed by a number and a message. The error type is indicated by a letter:

- I information
- E error
- F fatal error
- S internal compiler error
- W warning

Frontend

- F 1 evaluation expired

Your product evaluation period has expired. Contact your local TASKING office for the official product.

- W 2 unrecognized option: '*option*'

The option you specified does not exist. Check the invocation syntax for the correct option.

- E 4 expected *number* more '#endif'

The preprocessor part of the compiler found the '#if', '#ifdef' or '#ifndef' directive but did not find a corresponding '#endif' in the same source file. Check your source file that each '#if', '#ifdef' or '#ifndef' has a corresponding '#endif'.

- E 5 no source modules

You must specify at least one source file to compile.

- F 6 cannot create "*file*"

The output file or temporary file could not be created. Check if you have sufficient disk space and if you have write permissions in the specified directory.

- F 7 cannot open "*file*"

Check if the file you specified really exists. Maybe you misspelled the name, or the file is in another directory.

- F 8 attempt to overwrite input file "*file*"

The output file must have a different name than the input file.

E 9 unterminated constant character or string

This error can occur when you specify a string without a closing double-quote (") or when you specify a character constant without a closing single-quote ('). This error message is often preceded by one or more E 19 error messages.

F 11 file stack overflow

This error occurs if the maximum nesting depth (50) of file inclusion is reached. Check for #include files that contain other #include files. Try to split the nested files into simpler files.

F 12 memory allocation error

All free space has been used. Free up some memory by removing any resident programs, divide the file into several smaller source files, break expressions into smaller subexpressions or put in more memory.

W 13 prototype after forward call or old style declaration – ignored

Check that a prototype for each function is present before the actual call.

E 14 ';' inserted

An expression statement needs a semicolon. For example, after ++i in { int i; ++i }.

E 15 missing filename after -o option

The **-o** option must be followed by an output filename.

E 16 bad numerical constant

A constant must conform to its syntax. For example, 08 violates the octal digit syntax. Also, a constant may not be too large to be represented in the type to which it was assigned. For example, `int i = 0x1234567890;` is too large to fit in an integer.

E 17 string too long

This error occurs if the maximum string size (1500) is reached. Reduce the size of the string.

E 18 illegal character (*0xbexnumber*)

The character with the hexadecimal ASCII value *0xbexnumber* is not allowed here. For example, the '#' character, with hexadecimal value 0x23, to be used as a preprocessor command, may not be preceded by non-white space characters. The following is an example of this error:

```
char *s = #S ;      // error
```

E 19 newline character in constant

The newline character can appear in a character constant or string constant only when it is preceded by a backslash (\). To break a string that is on two lines in the source file, do one of the following:

- End the first line with the line-continuation character, a backslash (\).
- Close the string on the first line with a double quotation mark, and open the string on the next line with another quotation mark.

E 20 empty character constant

A character constant must contain exactly one character. Empty character constants (' ') are not allowed.

E 21 character constant overflow

A character constant must contain exactly one character. Note that an escape sequence (for example, \t for tab) is converted to a single character.

E 22 '#define' without valid identifier

You have to supply an identifier after a '#define'.

E 23 '#else' without '#if'

'#else' can only be used within a corresponding '#if', '#ifdef' or '#ifndef' construct. Make sure that there is a '#if', '#ifdef' or '#ifndef' statement in effect before this statement.

E 24 '#endif' without matching '#if'

'#endif' appeared without a matching '#if', '#ifdef' or '#ifndef' preprocessor directive. Make sure that there is a matching '#endif' for each '#if', '#ifdef' and '#ifndef' statement.

E 25 missing or zero line number

'#line' requires a non-zero line number specification.

E 26 undefined control

A control line (line with a '#*identifier*') must contain one of the known preprocessor directives.

W 27 unexpected text after control

'#ifdef' and '#ifndef' require only one identifier. Also, '#else' and '#endif' only have a newline. '#undef' requires exactly one identifier.

W 28 empty program

The source file must contain at least one external definition. A source file with nothing but comments is considered an empty program.

E 29 bad '#include' syntax

A '#include' must be followed by a valid header name syntax. For example, `#include <stdio.h` misses the closing '>'.

E 30 include file "file" not found

Be sure you have specified an existing include file after a '#include' directive. Make sure you have specified the correct path for the file.

E 31 end-of-file encountered inside comment

The compiler found the end of a file while scanning a comment. Probably a comment was not terminated. Do not forget a closing comment `*/` when using ANSI-C style comments.

E 32 argument mismatch for macro "name"

The number of arguments in invocation of a function-like macro must agree with the number of parameters in the definition. Also, invocation of a function-like macro requires a terminating `)` token. The following are examples of this error:

```
#define A(a) 1
int i = A(1,2);      /* error */

#define B(b) 1
int j = B(1;         /* error */
```

E 33 "name" redefined

The given identifier was defined more than once, or a subsequent declaration differed from a previous one. The following examples generate this error:

```
int i;
char i;                /* error */
main()
{
}
```

```
main()
{
    int j;
    int j;      /* error */
}
```

W 34 illegal redefinition of macro "*name*"

A macro can be redefined only if the body of the redefined macro is exactly the same as the body of the originally defined macro.

This warning can be caused by defining a macro on the command line and in the source with a '#define' directive. It also can be caused by macros imported from include files. To eliminate the warning, either remove one of the definitions or use an '#undef' directive before the second definition.

E 35 bad filename in '#line'

The string literal of a #line (if present) may not be a "wide-char" string. So, #line 9999 L"t45.c" is not allowed.

W 36 'debug' facility not installed

'#pragma debug' is only allowed in the debug version of the compiler.

W 37 attempt to divide by zero

A divide or modulo by zero was found. Adjust the expression or test if the second operand of a divide or modulo is zero.

E 38 non integral switch expression

A switch condition expression must evaluate to an integral value. So, char *p = 0; switch (p) is not allowed.

F 39 unknown error number: *number*

This error may not occur. If it does, contact your local TASKING office and provide them with the exact error message.

W 40 non-standard escape sequence

Check the spelling of your escape sequence (a backslash, \, followed by a number or letter), it contains an illegal escape character. For example, \c causes this warning.

- E 41 `#elif` without `#if`
The `#elif` directive did not appear within an `#if`, `#ifdef` or `#ifndef` construct. Make sure that there is a corresponding `#if`, `#ifdef` or `#ifndef` statement in effect before this statement.
- E 42 syntax error, expecting parameter type/declaration/statement
A syntax error occurred in a parameter list a declaration or a statement. This can have many causes, such as, errors in syntax of numbers, usage of reserved words, operator errors, missing parameter types, missing tokens.
- E 43 unrecoverable syntax error, skipping to end of file
The compiler found an error from which it could not recover. This error is in most cases preceded by another error. Usually, error E 42.
- I 44 in initializer "*name*"
Informational message when checking for a proper constant initializer.
- E 46 cannot hold that many operands
The value stack may not exceed 20 operands.
- E 47 missing operator
An operator was expected in the expression.
- E 48 missing right parenthesis
)' was expected.
- W 49 attempt to divide by zero – potential run-time error
An expression with a divide or modulo by zero was found. Adjust the expression or test if the second operand of a divide or modulo is zero.
- E 50 missing left parenthesis
(was expected.
- E 51 cannot hold that many operators
The state stack may not exceed 20 operators.
- E 52 missing operand
An operand was expected.

- E 53 missing identifier after 'defined' operator

An identifier is required in a `#if defined(identifier)`.

- E 54 non scalar controlling expression

Iteration conditions and 'if' conditions must have a scalar type (not a struct, union or a pointer). For example, after `static struct {int i;} si = {0};` it is not allowed to specify `while (si) ++si.i;`.

- E 55 operand has not integer type

The operand of a `'#if'` directive must evaluate to an integral constant. So, `#if 1.` is not allowed.

- W 56 '`<debugoption><level>`' no associated action

This warning can only appear in the debug version of the compiler. There is no associated debug action with the specified debug option and level.

- W 58 invalid warning number: *number*

The warning number you supplied to the `-w` option does not exist. Replace it with the correct number.

- F 59 sorry, more than *number* errors

Compilation stops if there are more than 40 errors.

- E 60 label "*label*" multiple defined

A label can be defined only once in the same function. The following is an example of this error:

```
f()  
{  
  lab1:  
  lab1:          /* error */  
}
```

- E 61 type clash

The compiler found conflicting types. For example, a long is only allowed on int or double, no specifiers are allowed with struct, union or enum. The following is an example of this error:

```
unsigned signed int i;      /* error */
```


E 62 bad storage class for "*name*"

The storage class specifiers `auto` and `register` may not appear in declaration specifiers of external definitions. Also, the only storage class specifier allowed in a parameter declaration is `register`.

E 63 "*name*" redeclared

The specified identifier was already declared. The compiler uses the second declaration. The following is an example of this error:

```
struct T { int i; };
struct T { long j; };      /* error */
```

E 64 incompatible redeclaration of "*name*"

The specified identifier was already declared. All declarations in the same function or module that refer to the same object or function must specify compatible types. The following is an example of this error:

```
f()
{
    int i;
    char i;      /* error */
}
```

W 66 function "*name*": variable "*name*" not used

A variable is declared which is never used. You can remove this unused variable or you can use the **-w66** option to suppress this warning.

W 67 illegal suboption: *option*

The suboption is not valid for this option. Check the invocation syntax for a list of all available suboptions.

W 68 function "*name*": parameter "*name*" not used

A function parameter is declared which is never used. You can remove this unused parameter or you can use the **-w68** option to suppress this warning.

E 69 declaration contains more than one basic type specifier

Type specifiers may not be repeated. The following is an example of this error:

```
int char i;      /* error */
```

E 70 'break' outside loop or switch

A `break` statement may only appear in a `switch` or a loop (`do`, `for` or `while`). So, `if (0) break;` is not allowed.

E 71 illegal type specified

The type you specified is not allowed in this context. For example, you cannot use the type `void` to declare a variable. The following is an example of this error:

```
void i;      /* error */
```

W 72 duplicate type modifier

Type qualifiers may not be repeated in a specifier list or qualifier list. The following is an example of this warning:

```
{ long long i; }      /* error */
```

E 73 object cannot be bound to multiple memories

Use only one memory attribute per object. For example, specifying both `rom` and `ram` to the same object is not allowed.

E 74 declaration contains more than one class specifier

A declaration may contain at most one storage class specifier. So, `register auto i;` is not allowed.

E 75 'continue' outside a loop

`continue` may only appear in a loop body (`do`, `for` or `while`). So, `switch (i) {default: continue;}` is not allowed.

E 76 duplicate macro parameter "*name*"

The given identifier was used more than one in the `formatl` parameter list of a macro definition. Each macro parameter must be uniquely declared.

E 77 parameter list should be empty

An identifier list, not part of a function definition, must be empty. For example, `int f (i, j, k);` is not allowed on declaration level.

E 78 'void' should be the only parameter

Within a function prototype of a function that does not except any arguments, `void` may be the only parameter. So, `int f(void, int);` is not allowed.

- E 79 constant expression expected

A constant expression may not contain a comma. Also, the bit field width, an expression that defines an enum, array-bound constants and switch case expressions must all be integral constant expressions.

- E 80 `#` operator shall be followed by macro parameter

The `#` operator must be followed by a macro argument.

- E 81 `##` operator shall not occur at beginning or end of a macro

The `##` (token concatenation) operator is used to paste together adjacent preprocessor tokens, so it cannot be used at the beginning or end of a macro body.

- W 86 escape character truncated to 8 bit value

The value of a hexadecimal escape sequence (a backslash, `\`, followed by a `'x'` and a number) must fit in 8 bits storage. The number of bits per character may not be greater than 8. The following is an example of this warning:

```
char c = '\xabc';      /* error */
```

- E 87 concatenated string too long

The resulting string was longer than the limit of 1500 characters.

- W 88 *"name"* redeclared with different linkage

The specified identifier was already declared. This warning is issued when you try to redeclare an object with a different basic storage class, and both objects are not declared extern or static. The following is an example of this warning:

```
int i;
int i();      /* error E 64 and warning */
```

- E 89 illegal bitfield declarator

A bit field may only be declared as an integer, not as a pointer or a function for example. So, `struct {int *a:1;} s;` is not allowed.

- E 90 *#error message*

The *message* is the descriptive text supplied in a `#error` preprocessor directive.

W 91 no prototype for function "*name*"

Each function should have a valid function prototype.

W 92 no prototype for indirect function call

Each function should have a valid function prototype.

I 94 hiding earlier one

Additional message which is preceded by error E 63. The second declaration will be used.

F 95 protection error: *message*

Something went wrong with the protection key initialization. The message could be: "Key is not present or printer is not correct.", "Can't read key.", "Can't initialize key.", or "Can't set key-model".

E 96 syntax error in `#define`

`#define id(` requires a right-parenthesis `)`'.

E 97 "... " incompatible with old-style prototype

If one function has a parameter type list and another function, with the same name, is an old-style declaration, the parameter list may not have ellipsis. The following is an example of this error:

```
int f(int, ...);
int f();          /* error, old-style */
```

E 98 function type cannot be inherited from a typedef

A `typedef` cannot be used for a function definition. The following is an example of this error:

```
typedef int INTFN();
INTFN f {return (0);}    /* error */
```

F 99 conditional directives nested too deep

`'#if`, `'#ifdef` or `'#ifndef` directives may not be nested deeper than 50 levels.

E 100 case or default label not inside switch

The `case:` or `default:` label may only appear inside a `switch`.

E 101 vacuous declaration

Something is missing in the declaration. The declaration could be empty or an incomplete statement was found. You must declare array declarators and `struct`, `union`, or `enum` members. The following are examples of this error:

```
int ;                                /* error */

static int a[2] = { };              /* error */
```

E 102 duplicate case or default label

Switch case values must be distinct after evaluation and there may be at most one default: label inside a switch.

E 103 may not subtract pointer from scalar

The only operands allowed on subtraction of pointers is pointer – pointer, or pointer – scalar. So, scalar – pointer is not allowed. The following is an example of this error:

```
int i;
int *pi = &i;
ff(1 - pi);                          /* error */
```

E 104 left operand of *operator* has not struct/union type

The first operand of a `'.'` or `'->'` must have a `struct` or `union` type.

E 105 zero or negative array size – ignored

Array bound constants must be greater than zero. So, `char a[0];` is not allowed.

E 106 different constructors

Compatible function types with parameter type lists must agree in number of parameters and in use of ellipsis. Also, the corresponding parameters must have compatible types. This error is usually followed by informational message I 111. The following is an example of this error:

```
int f(int);
int f(int, int);                      /* error different
                                     parameter list */
```

E 107 different array sizes

Corresponding array parameters of compatible function types must have the same size. This error is usually followed by informational message I 111. The following is an example of this error:

```
int f(int[][2]);  
int f(int[][3]);      /* error */
```

E 108 different types

Corresponding parameters must have compatible types and the type of each prototype parameter must be compatible with the widened definition parameter. This error is usually followed by informational message I 111. The following is an example of this error:

```
int f(int);  
int f(long);          /* error different type  
                        in parameter list */
```

E 109 floating point constant out of valid range

A floating point constant must have a value that fits in the type to which it was assigned. See section *Data Types* for the valid range of a floating point constant. The following is an example of this error:

```
float d = 10E9999;     /* error, too big */
```

E 110 function cannot return arrays or functions

A function may not have a return type that is of type array or function. A pointer to a function is allowed. The following are examples of this error:

```
typedef int F(); F f();      /* error */  
  
typedef int A[2]; A g();     /* error */
```

I 111 parameter list does not match earlier prototype

Check the parameter list or adjust the prototype. The number and type of parameters must match. This message is preceded by error E 106, E 107 or E 108.

E 112 parameter declaration must include identifier

If the declarator is a prototype, the declaration of each parameter must include an identifier. Also, an identifier declared as a `typedef` name cannot be a parameter name. The following are examples of this error:

```
int f(int g, int) {return (g);} /* error */

typedef int int_type;
int h(int_type) {return (0);} /* error */
```

E 114 incomplete struct/union type

The struct or union type must be known before you can use it. The following is an example of this error:

```
extern struct unknown sa, sb;
sa = sb; /* 'unknown' does not have a
          defined type */
```

The left side of an assignment (the lvalue) must be modifiable.

E 115 label "*name*" undefined

A goto statement was found, but the specified label did not exist in the same function or module. The following is an example of this error:

```
f1() { a: ; } /* W 116 */
f2() { goto a; } /* error, label 'a:' is
                  not defined in f2() */
```

W 116 label "*name*" not referenced

The given label was defined but never referenced. The reference of the label must be within the same function or module. The following is an example of this warning:

```
f() { a: ; } /* 'a' is not referenced */
```

E 117 "*name*" undefined

The specified identifier was not defined. A variable's type must be specified in a declaration before it can be used. This error can also be the result of a previous error. The following is an example of this error:

```
unknown i; /* error, 'unknown' undefined */
i = 1; /* as a result, 'i' is also
        undefined */
```

W 118 constant expression out of valid range

A constant expression used in a case label may not be too large. Also when converting a floating point value to an integer, the floating point constant may not be too large. This warning is usually preceded by error E 16 or E 109. The following is an example of this warning:

```
int i = 10E88; /* error and warning */
```

E 119 cannot take 'sizeof' bitfield or void type

The size of a bit field or void type is not known. So, the size of it cannot be taken.

E 120 cannot take 'sizeof' function

The size of a function is not known. So, the size of it cannot be taken.

E 121 not a function declarator

This is not a valid function. This may be due to a previous error. The following is an example of this error:

```
int f() return 0; /* missing '{ }' */
int g() { }      /* error, 'g' is not a formal
                  parameter and therefore,
                  this is not a valid function
                  declaration */
```

E 122 unnamed formal parameter

The parameter must have a valid name.

W 123 function should return something

A return in a non-void function must have an expression.

E 124 array cannot hold functions

An array of functions is not allowed.

E 125 function cannot return anything

A return with an expression may not appear in a void function.

W 126 missing return (function "*name*")

A non-void function with a non-empty function body must have a return statement.

E 129 cannot initialize "*name*"

Declarators in the declarator list may not contain initializations. Also, an extern declaration may have no initializer. The following are examples of this error:

```
{ extern int i = 0; } /* error */
int f( i ) int i=0;  /* error */
```


- W 130 operands of *operator* are pointers to different types

Pointer operands of an operator or assignment ('='), must have the same type. For example, the following code generates this warning:

```
long *pl;
int *pi = 0;
pl = pi;      /* warning */
```

- E 131 bad operand type(s) of *operator*

The operator needs an operand of another type. The following is an example of this error:

```
int *pi;
pi += 1.;      /* error, pointer on left; needs
                integral value on right */
```

- W 132 value of variable "*name*" is undefined

This warning occurs if a variable is used before it is defined. For example, the following code generates this warning:

```
int a,b;
a = b; /* warning, value of b unknown */
```

- E 133 illegal struct/union member type

A function cannot be a member of a struct or union. Also, bit fields may only have type int or unsigned.

- E 134 bitfield size out of range – set to 1

The bit field width may not be greater than the number of bits in the type and may not be negative. The following example generates this error:

```
struct i { unsigned i : 999; }; /* error */
```

- W 135 statement not reached

The specified statement will never be executed. This is for example the case when statements are present after a return.

- E 138 illegal function call

You cannot perform a function call on an object that is not a function. The following example generates this error:

```
int i, j;
j = i();      /* error, i is not a function */
```

E 139 *operator* cannot have aggregate type

The type name in a (cast) must be a scalar (not a struct, union or a pointer) and also the operand of a (cast) must be a scalar. The following are examples of this error:

```
static union ui {int a;} ui ;
ui = (union ui)9;          /* cannot cast to union */
ff( (int)ui );             /* cannot cast a union
                             to something else */
```

E 140 *type* cannot be applied to a register/bit/bitfield object or builtin/inline function

For example, the '&' operator (address) cannot be used on registers and bit fields. So, `func(&r6);` and `func(&bitf.a);` are invalid.

E 141 *operator* requires modifiable lvalue

The operand of the '++', or '--' operator and the left operand of an assignment or compound assignment (lvalue) must be modifiable. The following is an example of this error:

```
const int i = 1;
i = 3;          /* error, const cannot be
                  modified */
```

E 143 too many initializers

There may be no more initializers than there are objects. The following is an example of this error:

```
static int a[1] = {1, 2};      /* error,
                                only one object can be initialized */
```

W 144 enumerator "*name*" value out of range

An enum constant exceeded the limit for an int. The following is an example of this warning:

```
enum { A = INT_MAX, B };      /* warning,
                                B does not fit in an int anymore */
```

E 145 requires enclosing curly braces

A complex initializer needs enclosing curly braces. For example, `int a[] = 2;` is not valid, but `int a[] = {2};` is.

E 146 argument *#number*: memory spaces do not match

With prototypes, the memory spaces of arguments must match.

- W 147 argument *#number*: different levels of indirection

With prototypes, the types of arguments must be assignment compatible. The following code generates this warning:

```
int i; void func(int,int);
func( 1, &i );      /* warning, argument 2 */
```

- W 148 argument *#number*: struct/union type does not match

With prototypes, both the prototyped function argument and the actual argument was a struct or union., but they have different tags. The tag types should match. The following is an example of this warning:

```
f(struct s); /* prototype */
main()
{
    struct { int i; } t;
    f( t ); /* t has other type than s */
}
```

- E 149 object *"name"* has zero size

A struct or union may not have a member with an incomplete type. The following is an example of this error:

```
struct { struct unknown m; } s; /* error */
```

- W 150 argument *#number*: pointers to different types

With prototypes, the pointer types of arguments must be compatible. The following example generates this warning:

```
int f(int*);
long *l;
f(l);      /* warning */
```

- W 151 ignoring memory specifier

Memory specifiers for a struct, union or enum are ignored.

- E 152 operands of *operator* are not pointing to the same memory space

Be sure the operands point to the same memory space. This error occurs, for example, when you try to assign a pointer to a pointer from a different memory space.

E 153 `'sizeof' zero sized object`

An implicit or explicit `sizeof` operation references an object with an unknown size. This error is usually preceded by error E 119 or E 120, cannot take `'sizeof'`.

E 154 argument *#number*: struct/union mismatch

With prototypes, only one of the prototyped function argument or the actual argument was a `struct` or union. The types should match. The following is an example of this error:

```
f(struct s); /* prototype */

main()
{
    int i;
    f( i ); /* i is not a struct */
}
```

E 155 casting lvalue *'type'* to *'type'* is not allowed

The operand of the `'++'`, or `'--'` operator or the left operand of an assignment or compound assignment (lvalue) may not be cast to another type. The following is an example of this error:

```
int i = 3;
++(unsigned)i;      /* error, cast expression
                     is not an lvalue */
```

E 157 *"name"* is not a formal parameter

If a declarator has an identifier list, only its identifiers may appear in the declarator list. The following is an example of this error:

```
int f( i ) int a;    /* error */
```

E 158 right side of *operator* is not a member of the designated struct/union

The second operand of `'.'` or `'->'` must be a member of the designated struct or union.

E 160 pointer mismatch at *operator*

Both operands of *operator* must be a valid pointer. The following example generates this error:

```
int *pi = 44; /* right side not a pointer */
```

- E 161 aggregates around *operator* do not match

The contents of the structs, unions or arrays on both sides of the *operator* must be the same. The following example causes this error:

```
struct {int a; int b;} s;
struct {int c; int d; int e;} t;
s = t;          /* error */
```

- E 162 *operator* requires an lvalue or function designator

The '&' (address) operator requires an lvalue or function designator. The following is an example of this error:

```
int i;
i = &( i = 0 );
```

- W 163 operands of *operator* have different level of indirection

The types of pointers or addresses of the operator must be assignment compatible. The following is an example of this warning:

```
char **a;
char *b;
a = b;          /* warning */
```

- E 164 operands of *operator* may not have type 'pointer to void'

The operands of *operator* may not have operand (void *).

- W 165 operands of *operator* are incompatible: pointer vs. pointer to array

The types of pointers or addresses of the operator must be assignment compatible. A pointer cannot be assigned to a pointer to array. The following is an example of this warning:

```
main()
{
    typedef int array[10];
    array a;
    array *ap = a;      /* warning */
}
```

- E 166 *operator* cannot make something out of nothing

Casting type void to something else is not allowed. The following example generates this error:

```
void f(void);
main()
{
    int i;

    i = (int)f();    /* error */
}
```

E 170 recursive expansion of inline function "*name*"

An `_inline` function may not be recursive. The following example generates this error:

```
_inline int a (int i)
{
    a(i);    /* recursive call */
    return i;
}
main()
{
    a(1);    /* error */
}
```

E 171 too much tail-recursion in inline function "*name*"

If the function level is greater than or equal to 40 this error is given. The following example generates this error:

```
_inline void a ()
{
    a();
}
main()
{
    a();
}
```

W 172 adjacent strings have different types

When concatenating two strings, they must have the same type. The following example generates this warning:

```
char b[] = L"abc""def";    /* strings have
                             different types */
```

E 173 'void' function argument

A function may not have an argument with type `void`.

E 174 not an address constant

A constant address was expected. Unlike a static variable, an automatic variable does not have a fixed memory location and therefore, the address of an automatic is not a constant. The following is an example of this error:

```
int *a;
static int *b = a; /* error */
```

E 175 not an arithmetic constant

In a constant expression no assignment operators, no '++' operator, no '—' operator and no functions are allowed. The following is an example of this error:

```
int a;
static int b = a++; /* error */
```

E 176 address of automatic is not a constant

Unlike a static variable, an automatic variable does not have a fixed memory location and therefore, the address of an automatic is not a constant. The following is an example of this error:

```
int a;          /* automatic */
static int *b = &a; /* error */
```

W 177 static variable "*name*" not used

A static variable is declared which is never used. To eliminate this warning remove the unused variable.

W 178 static function "*name*" not used

A static function is declared which is never called. To eliminate this warning remove the unused function.

E 179 inline function "*name*" is not defined

Possibly only the prototype of the inline function was present, but the actual inline function was not. The following is an example of this error:

```

    _inline int a(void);      /* prototype */

main()
{
    int b;
    b = a();                  /* error */
};

```

- E 180 illegal target memory (*memory*) for pointer
The pointer may not point to *memory*. For example, a pointer to bitaddressable memory is not allowed.
- E 181 invalid cast to function
A cast to type function is not allowed. A cast to a function pointer type is allowed.
- W 182 argument *#number*: different types
With prototypes, the types of arguments must be compatible.
- W 183 variable '*name*' possibly uninitialized
Possibly an initialization statement is not reached, while a function should return something. The following is an example of this warning:

```

int a;

int f(void)
{
    int i;

    if ( a )
    {
        i = 0; /* statement not reached */
    }
    return i; /* warning */
}

```

- W 184 empty pragma name in *-z* option – ignored
The *-z* option requires a pragma name.
- I 185 (prototype synthesized at line *number* in "*name*")
This is an informational message containing the source file position where an old-style prototype was synthesized. This message is preceded by error E 146, W 147, W 148, W 150, E 154, W 182 or E 203.

- E 186 array of type bit is not allowed
An array cannot contain bit type variables.
- E 187 illegal structure definition
A structure can only be defined (initialized) if its members are known. So, `struct unknown s = { 0 };` is not allowed.
- E 188 structure containing bit-type fields is forced into bitaddressable area
This error occurs when you use a bitaddressable storage type for a structure containing bit-type members.
- E 189 pointer is forced to bitaddressable, pointer to bitaddressable is illegal
A pointer to bitaddressable memory is not allowed.
- W 190 "long float" changed to "float"
In ANSI C floating point constants are treated having type `double`, unless the constant has the suffix `'f'`. If you have specified an option to use float constants, a long floating point constant such as `123.12f1` is changed to a `float`.
- E 191 recursive struct/union definition
A `struct` or `union` cannot contain itself. The following example generates this error:
- ```
struct s { struct s a; } b; /* error */
```
- E 192 missing filename after `-f` option  
The `-f` option requires a filename argument.
- E 194 cannot initialize typedef  
You cannot assign a value to a typedef variable. So, `typedef i=2;` is not allowed.
- W 195 constant expression out of range -- truncated  
The resulting constant expression is too large to fit in the specified data type. The value is truncated. The following example generates this warning:

```
int i = 140000L; /* warning, value is too large
 to fit in an int */
```

- W 196    constant expression out of range due to signed/unsigned type mismatch

The resulting constant expression is too large to fit in the specified data type. The following example generates this warning:

```
int i = 40000U; /* the unsigned value is too large
 to fit in a signed int */
/* unsigned int i = 40000U; is OK */
```

- W 197    unrecognized -w argument: *argument*

The **-w** option only accepts a warning number or the text 'strict' as an argument. See the description of the **-w** option for details.

- W 198    trigraph sequence replaced

Trigraphs are used in the C language to create special characters on obsolete terminals with a limited character set. When they are replaced in your source, e.g. in a string, they may give rise to very obscure errors.

- F 199    demonstration package limits exceeded

The demonstration package has certain limits which are not present in the full version. Contact TASKING for a full version.

- W 200    unknown pragma "*name*" – ignored

The compiler ignores pragmas that are not known. For example, `#pragma unknown`.

- W 201    *name* cannot have storage type – ignored

A `register` variable or an automatic/parameter cannot have a storage type. To eliminate this warning, remove the storage type or place the variable outside a function.

- E 202    '*name*' is declared with 'void' parameter list

You cannot call a function with an argument when the function does not accept any (void parameter list). The following is an example of this error:

```

int f(void); /* void parameter list */

main()
{
 int i;
 i = f(i); /* error */
 i = f(); /* OK */
}

```

- E 203 too many/few actual parameters

With prototyping, the number of arguments of a function must agree with the prototype of the function. The following is an example of this error:

```

int f(int); /* one parameter */

main()
{
 int i;
 i = f(i,i); /* error, one too many */
 i = f(i); /* OK */
}

```

- W 204 U suffix not allowed on floating constant – ignored

A floating point constant cannot have a 'U' or 'u' suffix.

- W 205 F suffix not allowed on integer constant – ignored

An integer constant cannot have a 'F' or 'f' suffix.

- E 206 '*name*' named bit-field cannot have 0 width

A bit field must be an integral constant expression with a value greater than zero.

- W 208 unsupported MISRA C rule number %d.

Specified MISRA C rule number is not supported.

- E 209 +MISRA C rule %d violation: %s

A specified MISRA C rule is violated.

- E 212 "*name*": missing static function definition

A function with a `static` prototype misses its definition.

W 213 invalid string/character constant in non-active part of source  
This part of the source is skipped.

E 214 second occurrence of `#pragma asm` or `asm_noflush`  
`#pragma asm/#pragma endasm` blocks cannot be nested. Use  
`#pragma endasm` before starting a new `#pragma asm/#pragma`  
`endasm` block.

E 215 `"#pragma endasm"` without a `"#pragma asm"`  
A `#pragma endasm` must always have a corresponding `#pragma asm`  
or `#pragma asm_noflush`.

W 303 variable '*name*' uninitialized

Possibly an initialization statement is not reached, while a function  
should return something. The following is an example of this warning:

```
int a;

int f(void)
{
 int i;

 if (a)
 {
 i = 0; /* statement not reached */
 }
 return i; /* warning */
}
```

E 327 too many arguments to pass in registers for `_asmfunc` '*name*'  
An `_asmfunc` function uses a fixed register-based interface between C  
and assembly, but the number of arguments that can be passed is  
limited by the number of available registers. With function *name* this  
limit was reached.

**Backend**

W 501 function qualifier used on non-function

A function qualifier can only be used on functions.

W 502 `_fract` constant saturation occurred

An overflow occurred. The following is an example of this warning:

```
_fract a;

int f(void)
{
 a = .75 + .5 /* 1.25, overflow */
}
```

E 503 conversion between `_fract` types and integer not allowed

E 504 initialization with constant not allowed for circular pointers

E 505 no conversions allowed to/from packed types

Packed variables cannot be combined with any other type.

E 506 `_sat _packb` add/subtract is not supported by the TriCore instruction set

W 508 duplicate function qualifier

Only one function qualifier is allowed.

E 509 function definition not allowed for `_syscallfunc()` function "*name*"

The `_syscallfunc` function qualifier can only be used at a function declaration, not at a function definition.

E 510 pointer to `_syscallfunc()` function "*name*" not allowed

E 511 interrupt function must have void result and void parameter list

A function declared with `_interrupt(n)` may not accept any arguments and may not return anything.

W 512 illegal interrupt number '*number*' – ignored

The interrupt vector number must be in the range 0 to 255. Any other number is illegal.

E 513 calling an interrupt routine

An interrupt function cannot be called directly, you must use the intrinsic function `_enable()`.

- E 514    conflict in '`_interrupt/_stackparm`' attribute  
The attributes of the current function qualifier declaration and the previous function qualifier declaration are not the same.
- E 515    different '`_interrupt`' number  
The interrupt number of the current function qualifier declaration and the previous function qualifier declaration are not the same.
- E 516    '`memory_type`' is illegal memory for function  
The storage type is not valid for this function.
- E 517    `_packb/_packhw` division is not supported
- E 526    Identifier *name* unknown
- W 527    `_sat` ignored for '*operation*' operator on type `_accum`
- E 528    `_at()` requires a numerical address  
You can only use an expression that evaluates to a numerical address.
- E 530    `_at()/_atbit()` only valid for global variables  
Only global variables can be placed on absolute addresses.
- E 531    `_at()/_atbit()` only allowed for uninitialized variables  
Absolute variables cannot be initialized.
- E 532    `_atbit()` only valid for '`_at`' variables
- W 533    language extension keyword used as identifier  
A language extension keyword is a reserved word, and reserved words cannot be used as an identifier.
- W 534    `#pragma` section: duplicate section suffix ignored  
Only one section suffix is allowed after a `#pragma` section if you have not specified a section type.
- E 535    `#pragma` section: unknown section type  
See the description of `#pragma` section in paragraph *Pragmas* for a list of valid section types.
- E 536    `#pragma` section: '=' expected  
A section type must be followed by an '=' (equal) sign and a section name.

- E 537 `#pragma` section: section name expected  
A section type must be followed by an '=' (equal) sign and a section name.
- E 538 `#pragma` section: syntax error  
See paragraph *Pragmas* for a description of `#pragma` section.
- W 540 `#pragma` stack underflow  
This warning occurs if you use a `#pragma switch restore` while there were no options saved by a previous `#pragma switch`.
- W 541 `pragma` name should be followed by on/off/restore, a number or nothing  
Check the usage of the `pragma` for details.
- W 542 `#pragma optimize`: stack underflow  
This warning occurs if you use a `#pragma optimize restore` while there were no options saved by a previous `#pragma optimize`.
- W 543 `#pragma switch`: stack underflow  
This warning occurs if you use a `#pragma switch restore` while there were no options saved by a previous `#pragma switch`.
- W 544 `#pragma switch` must be followed by one of: auto, linear, jumptab, lookup, restore  
You have given no or a wrong argument to the `#pragma switch`.
- E 547 assignment of a (non-)circular object to a (non-)circular pointer
- E 549 only `_circ` pointers and arrays/structs/unions are allowed – ignored on struct/union member '*name*'  
`_circ` is only allowed on structures and unions if its member is of type `_circ`.
- F 550 file write error  
Check if there is enough free space left and check if you have write permissions.
- F 552 pointer to variable of type `-bit` is not allowed

- E 556 `_atbit()` only possible on objects, not on constant addresses  
Use `_atbit()` to define bit variables within variables with a previously defined name.
- E 558 bit position must be a constant value in the range [0..32)
- E 560 builtin function "*name*" requires constant argument  
Check the usage of the intrinsic (builtin) function for details.
- W 561 "enum" bitfield type treated as "int"  
The type enum is not allowed on bitfields, so it is treated as an int.
- E 562 `#pragma asm:` cannot use register *name*  
See section *Inline Assembly* for a list of the registers you can use.
- E 563 `#pragma asm:` scratch register index must be less than 10  
A scratch register must have a number in the range 0-9.
- E 564 `#pragma asm:` register *name* mentioned multiple times  
Use unique register names.
- E 565 `#pragma asm:` syntax error  
See section *Inline Assembly* for the syntax of `#pragma asm`.
- E 566 `#pragma endasm:` register *name* must be declared in `#pragma asm`  
  
The register you want to use after a `#pragma endasm` must be previously declared in the corresponding `#pragma asm`.
- E 567 `#pragma endasm:` type of register *name* does not agree with `#pragma asm`  
  
The scratch register you want to use after a `#pragma endasm` must be of the same type as previously declared in the corresponding `#pragma asm`.
- E 568 `#pragma endasm:` cannot assign register *name* to multiple variables  
  
A register can be assigned to a variable only once after a `#pragma endasm`.



E 569 `#pragma endasm`: syntax error

See section *Inline Assembly* for the syntax of `#pragma endasm`.

E 570 `#pragma asm`: unknown scratch register *name*

The scratch register you want to use within a `#pragma asm` part must be previously declared in the corresponding `#pragma asm`.

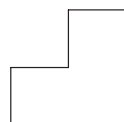
# CHAPTER 6

## LIBRARIES

---



**TASKING**



---

# 6

# CHAPTER

---

## 6.1 INTRODUCTION

This chapter describes the library functions delivered with the compiler. Some functions (e.g. `printf()`, `scanf()`) can be edited to match your needs. **ctri** come with libraries in object format and with header files containing the appropriate prototype of the library functions. The library functions are also shipped in source code (C or assembly).

A number of standard operations within C are too complex to generate inline code for. These operations are implemented as run-time library functions. The run-time library routines are added to the C library.

## 6.2 HEADER FILES

The following header files are delivered with the C compiler:

- <assert.h>** `assert`
- <ctype.h>** `isalnum`, `isalpha`, `isascii`, `isctrl`, `isdigit`, `isgraph`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `isxdigit`, `toascii`, `_tolower`, `tolower`, `_toupper`, `toupper`
- <errno.h>** Error numbers. No C functions.
- <fcntl.h>** Definition of flags used by `open()`.
- <float.h>** `copysign`, `isfinite`, `isinf`, `isnan`, `scalb`. Constants related to floating point arithmetic.
- <fss.h>** Definitions for file system simulation.
- <limits.h>** Limits and sizes of integral types. No C functions.
- <locale.h>** `localeconv`, `setlocale`. Delivered as skeletons.
- <malloc.h>** Non-ANSI C header file with prototypes of `malloc` and `free`.
- <math.h>** `acos`, `asin`, `atan`, `atan2`, `ceil`, `cos`, `cosh`, `exp`, `fabs`, `floor`, `fmod`, `frexp`, `ldexp`, `log`, `log10`, `modf`, `pow`, `sin`, `sinh`, `sqrt`, `tan`, `tanh`
- <setjmp.h>** `longjmp`, `setjmp`
- <signal.h>** `raise`, `signal`. Functions are delivered as skeletons.
- <stdarg.h>** `va_arg`, `va_end`, `va_start`

- <stddef.h>**    offsetof, definition of special types.
- <stdio.h>**    clearerr, \_close, fclose, feof, ferror, fflush, fgetc, fgetpos, fgets, fopen, fprintf, fputc, fputs, fread, freopen, fscanf, fseek, fsetpos, ftell, fwrite, getc, getchar, gets, \_lseek, \_open, perror, printf, putc, putchar, puts, \_read, remove, rename, rewind, scanf, setbuf, setvbuf, sprintf, sscanf, tmpfile, tmpnam, ungetc, vfprintf, vprintf, vsprintf, \_unlink, \_write
- <stdlib.h>**    abort, abs, atexit, atof, atoi, atol, bsearch, calloc, div, exit, free, getenv, labs, ldiv, malloc, mblen, mbstowcs, mbtowc, qsort, rand, realloc, srand, strtod, strtol, strtoul, system, wcstombs, wctomb
- <string.h>**    memchr, memcmp, memcpy, memmove, memset, strcat, strchr, strcmp, strcol, strcpy, strcspn, strerror, strlen, strncat, strncmp, strncpy, strpbrk, strchr, strspn, strstr, strtok, strxfrm
- <time.h>**    asctime, clock, ctime, difftime, gmtime, localtime, mktime, strftime, time. All functions are delivered as skeletons.
- <unistd.h>**    Non-ANSI C header file with prototypes for standard POSIX I/O functions. access, chdir, close, getcwd, lseek, read, stat, unlink, write.

### 6.3 C LIBRARIES

The C library contains C library functions. All C library functions are described in this chapter. These functions are only called by explicit function calls in your application program.

#### *Library directories*

- The standard set of libraries for TC1 derivatives is located in the system `lib\tc1` directory.
- The standard set of libraries for TC2 derivatives is located in the system `lib\tc2` directory.
- A protected library set for the TC112 CPU functional problems is located in the system `lib\p\tc112` directory.
- A protected library set for the TC113 CPU functional problems is located in the system `lib\p\tc113` directory.

The protected library sets provide software bypasses for all TC112 and TC113 supported CPU functional problems. They must be used in conjunction with the appropriate C compiler workarounds for CPU functional problems. For more details refer to Appendix C, *CPU Functional Problems*.

The C library uses the following name syntax:

| Library to link | Description                                                                           |
|-----------------|---------------------------------------------------------------------------------------|
| libc.a (def.)   | C library double precision                                                            |
| libcs.a         | C library single precision ( <b>-F</b> option)                                        |
| libcs_fpu.a     | C library single precision with FPU instructions ( <b>-F</b> and <b>-FPU</b> options) |
| libfpn.a (def.) | Floating point library (no trapping)                                                  |
| libfpt.a        | Floating point library (trapping)                                                     |
| libfpn_fpu.a    | Floating point library with FPU instructions (no trapping, <b>-FPU</b> option)        |
| libfpt_fpu.a    | Floating point library with FPU instructions trapping, <b>-FPU</b> option)            |

Table 6-1: C and floating point library name syntax



The **lktri** linker is using this naming convention when specifying the **-l** option. For example, with **-lc** the linker is looking for `libc.a` in the system `lib\tc1` directory. Specifying the libraries is a job taken care of by the control program.

When you use floating point, the floating point library must always be the last library linked, it should be placed after the C library. Arithmetic routines like `sin()`, `cos()`, etc. are not present in these libraries, they only contain basic floating point operations like ADD and MUL. These operations are only called implicitly in your application program.



The non-trapping floating point libraries `libfpn.a` and `libfpn_fpu.a` have been optimized for speed. As a result, they do not completely follow the IEEE-754 floating point standard for all exceptional cases.

### 6.3.1 SINGLE PRECISION FLOATING POINT

In ANSI C all mathematical functions (`<math.h>`), are based on `double` arguments and `double` return type. So, even if you are using only `float` variables in your code, the language definition dictates promotion to `double`, when using the math functions or floating point formatters (`printf()` and `scanf()`). The result is more code and less execution speed. In fact the ANSI approach introduces a performance penalty.

To improve the code size and execution speed, the compiler supports the option **-F** to force single precision floating point usage. If you use **-F**, a `float` variable passed as an argument is no longer promoted to `double` when calling a variable argument function or an old style K&R function, and the type `double` is treated as `float`. It is obvious that this affects the whole application (including libraries). Therefore, special single precision versions of the floating point libraries are delivered with the package. When using **-F**, these libraries must be used. It is not possible to mix C modules created with the **-F** option and C modules which are using the regular ANSI approach.

The **-Fc** option only treats floating point constants (having no suffix) as `float` instead of `double`.

### 6.3.2 C LIBRARY IMPLEMENTATION DETAILS

A detailed description of the delivered C library is shown in the following list.

Explanation :

- Y – Fully implemented
- I – Implemented via file system simulation
- L – Delivered as a skeleton

| File     | Imple-<br>mented | Routine name     | Description / Reason                                                                 |
|----------|------------------|------------------|--------------------------------------------------------------------------------------|
| assert.h | Y                | 'assert()' macro | Macro definition                                                                     |
| ctype.h  | Y                |                  | Most of the routines are delivered as macro AND as function (as prescribed by ANSI). |
|          | Y                | isalnum          |                                                                                      |
|          | Y                | isalpha          |                                                                                      |
|          | Y                | iscntrl          |                                                                                      |
|          | Y                | isdigit          |                                                                                      |
|          | Y                | isgraph          |                                                                                      |
|          | Y                | islower          |                                                                                      |
|          | Y                | isprint          |                                                                                      |
|          | Y                | ispunct          |                                                                                      |
|          | Y                | isspace          |                                                                                      |
|          | Y                | isupper          |                                                                                      |
|          | Y                | isxdigit         |                                                                                      |
|          | Y                | tolower          |                                                                                      |
|          | Y                | toupper          |                                                                                      |
|          | Y                | _tolower         | Not defined by ANSI                                                                  |
|          | Y                | _toupper         | Not defined by ANSI                                                                  |
|          | Y                | isascii          | Not defined by ANSI                                                                  |
|          | Y                | toascii          | Not defined by ANSI                                                                  |
| errno.h  | Y                |                  | Only Macros                                                                          |
| fcntl.h  | Y                |                  | Definitions of flags used by _open                                                   |
|          | I                | open             |                                                                                      |
| float.h  | Y                |                  |                                                                                      |
| limits.h | Y                |                  | Only Macros                                                                          |
| locale.h | Y                |                  |                                                                                      |
|          | L                | localeconv       | No OS present                                                                        |
|          | L                | setlocale        | No OS present                                                                        |



| File     | Imple-<br>mented                                                                                 | Routine name                                                                                                                                                                | Description / Reason |
|----------|--------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|
| math.h   | Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y | acos<br>asin<br>atan<br>atan2<br>ceil<br>cos<br>cosh<br>exp<br>fabs<br>floor<br>fmod<br>frexp<br>ldexp<br>log<br>log10<br>modf<br>pow<br>sin<br>sinh<br>sqrt<br>tan<br>tanh |                      |
| setjmp.h | Y<br>Y<br>Y                                                                                      | longjmp<br>setjmp                                                                                                                                                           |                      |
| signal.h | Y<br>Y<br>Y                                                                                      | raise<br>signal                                                                                                                                                             |                      |
| stdarg.h | Y<br>Y<br>Y<br>Y                                                                                 | va_arg<br>va_end<br>va_start                                                                                                                                                |                      |
| stddef.h | Y                                                                                                |                                                                                                                                                                             | Only Macros          |

| File    | Imple-<br>mented | Routine name | Description / Reason                                                  |
|---------|------------------|--------------|-----------------------------------------------------------------------|
| stdio.h | Y                |              |                                                                       |
|         | Y                | clearerr     |                                                                       |
|         | I                | fclose       |                                                                       |
|         | Y                | feof         |                                                                       |
|         | Y                | ferror       |                                                                       |
|         | I                | fflush       |                                                                       |
|         | I                | fgetc        |                                                                       |
|         | I                | fgetpos      |                                                                       |
|         | I                | fgets        |                                                                       |
|         | I                | fopen        |                                                                       |
|         | I                | fprintf      |                                                                       |
|         | I                | fputc        |                                                                       |
|         | I                | fputs        |                                                                       |
|         | I                | fread        |                                                                       |
|         | I                | freopen      |                                                                       |
|         | I                | fscanf       |                                                                       |
|         | I                | fseek        |                                                                       |
|         | I                | fsetpos      |                                                                       |
|         | I                | ftell        |                                                                       |
|         | I                | fwrite       |                                                                       |
|         | I                | getc         |                                                                       |
|         | I                | getchar      |                                                                       |
|         | I                | gets         |                                                                       |
|         | Y                | perror       |                                                                       |
|         | I                | printf       |                                                                       |
|         | I                | putc         |                                                                       |
|         | I                | putchar      |                                                                       |
|         | I                | puts         |                                                                       |
|         | L                | remove       |                                                                       |
|         | L                | rename       |                                                                       |
|         | I                | rewind       |                                                                       |
|         | I                | scanf        |                                                                       |
|         | Y                | setbuf       |                                                                       |
|         | Y                | setvbuf      |                                                                       |
|         | Y                | sprintf      |                                                                       |
|         | Y                | sscanf       |                                                                       |
|         | L                | tmpfile      |                                                                       |
|         | L                | tmpnam       | Delivered as a random name generator, but should use some process ID. |
|         | Y                | ungetc       |                                                                       |
|         | I                | vfprintf     |                                                                       |
|         | I                | vprintf      |                                                                       |
|         | Y                | vsprintf     |                                                                       |



| File     | Imple-<br>mented                                                                                                              | Routine name                                                                                                                                                                                                                                                          | Description / Reason                                                                                                                                                                                                                                                                                     |
|----------|-------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|          | I<br>I<br>I<br>I<br>I<br>I                                                                                                    | _close<br>_open<br>_lseek<br>_read<br>_unlink<br>_write                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                          |
| stdlib.h | Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>L<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br>Y<br><br>L<br>L<br>L<br>L<br>L | abort<br>abs<br>atexit<br>atof<br>atoi<br>atol<br>bsearch<br>calloc<br>div<br>exit<br>free<br>getenv<br>labs<br>ldiv<br>malloc<br>qsort<br>strtod<br>strtol<br>strtoul<br>rand<br>realloc<br>srand<br><br>system<br>mblen<br>mbstowcs<br>mbtowc<br>wcstombs<br>wctomb | Calls _exit() in cstart<br><br><br><br><br><br><br><br>Calls _exit() in cstart<br><br>No OS present<br><br><br><br><br><br><br><br><br><br><br>No OS present<br>wide chars not supported<br>wide chars not supported<br>wide chars not supported<br>wide chars not supported<br>wide chars not supported |

[illegible]

### 6.3.3 C LIBRARY INTERFACE DESCRIPTION

#### ***\_close***

```
#include <stdio.h>
int _close(int fd);
```

Low level file close function. `_close` is used by the functions `close` and `fclose`. The given file descriptor should be properly closed, any buffer is already flushed. This function interfaces to CrossView Pro's file system simulation.

#### ***\_lseek***

```
#include <stdio.h>
off_t _lseek(int fd, off_t offset, int whence);
```

Low level file positioning function. `_lseek` is used by all file positioning functions (`fgetpos`, `fseek`, `fsetpos`, `ftell`, `rewind`). This function interfaces to CrossView Pro's file system simulation.

#### ***\_open***

```
#include <stdio.h>
int _open(int fd, int flags);
```

Low level file open function. `_open` is used by the functions `fopen` and `freopen`. The given file descriptor should be properly opened. This function interfaces to CrossView Pro's file system simulation.

#### ***\_read***

```
#include <stdio.h>
size_t
_read(int fd, char *buffer, size_t count);
```

Low level input function. It reads a sequence of characters from a file. This function interfaces to CrossView Pro's file system simulation.

**Returns**      the number of characters read.

***\_tolower***

```
#include <ctype.h>
int _tolower(int c);
```

Converts *c* to a lowercase character, does not check if *c* really is an uppercase character. This is a non-ANSI function.

**Returns**      the converted character.

***\_toupper***

```
#include <ctype.h>
int _toupper(int c);
```

Converts *c* to an uppercase character, does not check if *c* really is a lowercase character. This is a non-ANSI function.

**Returns**      the converted character.

***\_unlink***

```
#include <stdio.h>
int _unlink(const char *name);
```

Low level file remove function. *\_unlink* is used by the function *remove*. This function interfaces to CrossView Pro's file system simulation.

***\_write***

```
#include <stdio.h>
size_t
_write(int fd, char *buffer, size_t count);
```

Low level output function. It writes a sequence of characters to a file. This function interfaces to CrossView Pro's file system simulation.

**Returns**      the number of characters correctly written.

***abort***

```
#include <stdlib.h>
void abort(void);
```

Terminates the program abnormally. It calls the function `_exit`, which is defined in the start-up module.

**Returns**      nothing.

***abs***

```
#include <stdlib.h>
int abs(int n);
```

**Returns**      the absolute value of the signed int argument.

***access***

```
#include <unistd.h>
int access(const char * name, int mode);
```

Use the file system simulation feature of CrossView Pro to check the permissions of a file on the host. mode specifies the type of access and is a bit pattern constructed by a logical OR of the following values:

|      |                                     |
|------|-------------------------------------|
| R_OK | Checks read permission.             |
| W_OK | Checks write permission.            |
| X_OK | Checks execute (search) permission. |
| F_OK | Checks to see if the file exists.   |

**Returns**      zero if successful,  
                 -1 on error.

***acos***

```
#include <math.h>
double acos(double x);
```

**Returns**      the arccosine  $\cos^{-1}(x)$  of  $x$  in the range  $[0, \pi]$ ,  
                  $x \in [-1, 1]$ .

***asctime***

```
#include <time.h>
char *asctime(const struct tm *tp);
```

Converts the time in the structure *\*tp* into a string of the form:

```
Mon Jan 21 16:15:14 1989\n\0
```

**Returns** the time in string form.

***asin***

```
#include <math.h>
double asin(double x);
```

**Returns** the arcsine  $\sin^{-1}(x)$  of *x* in the range  $[-\pi/2, \pi/2]$ ,  $x \in [-1, 1]$ .

***assert***

```
#include <assert.h>
void assert(int expr);
```

When compiled with NDEBUG, this is an empty macro. When compiled without NDEBUG defined, it checks if *expr* is true. If it is true, then a line like:

```
"Assertion failed: expression, file filename, line num"
```

is printed.

**Returns** nothing.

***atan***

```
#include <math.h>
double atan(double x);
```

**Returns** the arctangent  $\tan^{-1}(x)$  of *x* in the range  $[-\pi/2, \pi/2]$ ,  $x \in [-1, 1]$ .



***atan2***

```
#include <math.h>
double atan2(double y, double x);
```

**Returns** the result of:  $\tan^{-1}(y/x)$  in the range  $[-\pi, \pi]$ .

***atexit***

```
#include <stdlib.h>
int atexit(void (*fcn)(void));
```

Registers the function *fcn* to be called when the program terminates normally.

**Returns** zero, if program terminates normally.  
non-zero, if the registration cannot be made.

***atof***

```
#include <stdlib.h>
double atof(const char *s);
```

Converts the given string to a double value. White space is skipped, conversion is terminated at the first unrecognized character.

**Returns** the double value.

***atoi***

```
#include <stdlib.h>
int atoi(const char *s);
```

Converts the given string to an integer value. White space is skipped, conversion is terminated at the first unrecognized character.

**Returns** the integer value.

***atol***

```
#include <stdlib.h>
long atol(const char *s);
```

Converts the given string to a long value. White space is skipped, conversion is terminated at the first unrecognized character.

**Returns**      the long value.

***bsearch***

```
#include <stdlib.h>
void *bsearch(const void *key,
 const void *base, size_t n, size_t size, int (* cmp)
 (const void *, const void *));
```

This function searches in an array of *n* members, for the object pointed to by *ptr*. The initial base of the array is given by *base*. The size of each member is specified by *size*. The given array must be sorted in ascending order, according to the results of the function pointed to by *cmp*.

**Returns**      a pointer to the matching member in the array, or NULL when not found.

***calloc***

```
#include <stdlib.h>
void *calloc(size_t nobj, size_t size);
```

The allocated space is filled with zeros. The maximum space that can be allocated can be changed by customizing the heap size (see the section *Heap*). By default no heap is allocated. When "calloc()" is used while no heap is defined, the locator gives an error.

**Returns**      a pointer to space in external memory for *nobj* items of *size* bytes length.  
NULL if there is not enough space left.

***ceil***

```
#include <math.h>
double ceil(double x);
```

**Returns** the smallest integer not less than *x*, as a double.

***chdir***

```
#include <unistd.h>
int chdir(const char *path);
```

Use the file system simulation feature of CrossView Pro to change the current directory on the host to the directory indicated by *path*.

**Returns** zero if successful,  
-1 on error.

***clearerr***

```
#include <stdio.h>
void clearerr(FILE *stream);
```

Clears the end of file and error indicators for *stream*.

**Returns** nothing.

***clock***

```
#include <time.h>
clock_t clock(void);
```

Determines the processor time used.

**Returns** number of microseconds since the last reset, assuming a 100 MHz *cpu*.

***close***

```
#include <unistd.h>
int close(int fd);
```

File close function. The given file descriptor should be properly closed. This function calls `_close`.

**Returns**      zero if successful,  
                 -1 on error.

***copysign***

```
#include <float.h>
double copysign(double d, double sign);
```

IEEE-754-1985 recommended function. Copy the sign of the second argument to the value of the first argument and return that as result.

**Returns**      the first argument with the sign of the second argument.

***cos***

```
#include <math.h>
double cos(double x);
```

**Returns**      the cosine of  $x$ .

***cosh***

```
#include <math.h>
double cosh(double x);
```

**Returns**      the hyperbolic cosine of  $x$ .

***ctime***

```
#include <time.h>
char *ctime(const time_t *tp);
```

Converts the calendar time *\*tp* into local time, in string form. This function is the same as:

```
asctime(localtime(tp));
```

**Returns** the local time in string form.

***difftime***

```
#include <time.h>
double
difftime(time_t time2, time_t time1);
```

**Returns** the result of *time2 - time1* in seconds.

***div***

```
#include <stdlib.h>
div_t div(int num, int denom);
```

Both arguments are integers. The returned quotient and remainder are also integers.

**Returns** a structure containing the quotient and remainder of *num* divided by *denom*.

***exit***

```
#include <stdlib.h>
void exit(int status);
```

Terminates the program normally. Acts as if 'main()' returns with *status* as the return value.

**Returns** zero, on successful termination.

***exp***

```
#include <math.h>
double exp(double x);
```

**Returns** the result of the exponential function  $e^x$ .

***fabs***

```
#include <math.h>
double fabs(double x);
```

**Returns** the absolute double value of  $x$ .  $|x|$

***fclose***

```
#include <stdio.h>
int fclose(FILE *stream)
```

Flushes any unwritten data for `stream`, discards any unread buffered input, frees any automatically allocated buffer, then closes the `stream`.

**Returns** zero if the `stream` is successfully closed, or EOF on error.

***feof***

```
#include <stdio.h>
int feof(FILE *stream);
```

**Returns** a non-zero value if the end-of-file indicator for `stream` is set.

***ferror***

```
#include <stdio.h>
int ferror(FILE *stream);
```

**Returns** a non-zero value if the error indicator for `stream` is set.

***fflush***

```
#include <stdio.h>
int fflush(FILE *stream);
```

Writes any buffered but unwritten data, if *stream* is an output stream. If *stream* is an input stream, the effect is undefined.

**Returns**      zero if successful, or EOF on a write error.

***fgetc***

```
#include <stdio.h>
int fgetc(FILE *stream);
```

Reads one character from the given *stream*.

**Returns**      the read character, or EOF on error.

***fgetpos***

```
#include <stdio.h>
int fgetpos(FILE *stream, fpos_t *ptr);
```

Stores the current value of the file position indicator for the stream pointed to by *stream* in the object pointed to by *ptr*. The type *fpos\_t* is suitable for recording such values.

**Returns**      zero if successful,  
a non-zero value on error.

***fgets***

```
#include <stdio.h>
char *fgets(char *s, int n, FILE *stream);
```

Reads at most the next *n*-1 characters from the given stream into the array *s* until a newline is found.

**Returns**      *s*, or NULL on EOF or error.

***floor***

```
#include <math.h>
double floor(double x);
```

**Returns** the largest integer not greater than *x*, as a double.

***fmod***

```
#include <math.h>
double fmod(double x, double y);
```

**Returns** the floating-point remainder of *x/y*, with the same sign as *x*. If *y* is zero, the result is implementation-defined.

***fopen***

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
```

Opens a file for a given mode.

**Returns** a stream. If the file cannot not be opened, NULL is returned.

You can specify the following values for mode:

|      |                                                                                           |
|------|-------------------------------------------------------------------------------------------|
| "r"  | read; open text file for reading                                                          |
| "w"  | write; create text file for writing; if the file already exists its contents is discarded |
| "a"  | append; open existing text file or create new text file for writing at end of file        |
| "r+" | open text file for update; reading and writing                                            |
| "w+" | create text file for update; previous contents if any is discarded                        |
| "a+" | append; open or create text file for update, writes at end of file                        |



The update mode (with a '+') allows reading and writing of the same file. In this mode the function `fflush` must be called between a read and a write or vice versa. By including the letter `b` after the initial letter, you can indicate that the file is a binary file. E.g. `"rb"` means read binary, `"w+b"` means create binary file for update. The filename is limited to `FILENAME_MAX` characters. At most `FOPEN_MAX` files may be open at once.

### ***fprintf***

```
#include <stdio.h>
int fprintf(FILE *stream, const char *format, ...);
```

Performs a formatted write to the given stream.



See also `"printf()"`, `"_write()"`.

### ***fputc***

```
#include <stdio.h>
int fputc(int c, FILE *stream);
```

Puts one character onto the given stream.



See also `"_write()"`.

**Returns**      EOF on error.

### ***fputs***

```
#include <stdio.h>
int fputs(const char *s, FILE *stream);
```

Writes the string to a stream. The terminating NULL character is not written.



See also `"_write()"`.

**Returns**      NULL if successful, or EOF on error.

### ***fread***

```
#include <stdio.h>
size_t fread(void *ptr, size_t size,
 size_t nobj, FILE *stream);
```

Reads *nobj* members of *size* bytes from the given *stream* into the array pointed to by *ptr*.



See also "*\_read()*".

**Returns**      the number of successfully read objects.

### ***free***

```
#include <stdlib.h>
void free(void *p);
```

Deallocates the space pointed to by *p*. *p* Must point to space earlier allocated by a call to "*calloc()*", "*malloc()*" or "*realloc()*". Otherwise the behavior is undefined.



See also "*calloc()*", "*malloc()*" and "*realloc()*".

**Returns**      nothing

### ***freopen***

```
#include <stdio.h>
FILE * freopen(const char *filename,
 const char *mode, FILE *stream);
```

Opens a file for a given *mode* associates the *stream* with it. This function is normally used to change the files associated with *stdin*, *stdout*, or *stderr*.



See also "*fopen()*".

**Returns**      *stream*, or NULL on error.

***frexp***

```
#include <math.h>
double frexp(double x, int *exp);
```

Splits *x* into a normalized fraction in the interval  $[1/2, 1>$ , which is returned, and a power of 2, which is stored in *\*exp*. If *x* is zero, both parts of the result are zero. For example: `frexp( 4.0, &var )` results in  $0.5 \cdot 2^3$ . The function returns 0.5, and 3 is stored in *var*.

**Returns**      the normalized fraction.

***fscanf***

```
#include <stdio.h>
int fscanf(FILE *stream, const char *format, ...);
```

Performs a formatted read from the given stream.



See also "scanf()", "\_read()".

**Returns**      the number of items converted successfully.

***fseek***

```
#include <stdio.h>
int fseek(FILE *stream, long offset, int origin);
```

Sets the file position indicator for *stream*. A subsequent read or write will access data beginning at the new position. For a binary file, the position is set to *offset* characters from *origin*, which may be `SEEK_SET` for the beginning of the file, `SEEK_CUR` for the current position in the file, or `SEEK_END` for the end-of-file. For a text stream, *offset* must be zero, or a value returned by `ftell`. In this case *origin* must be `SEEK_SET`.

**Returns**      zero if successful,  
a non-zero value on error.

***fsetpos***

```
#include <stdio.h>
int fsetpos(FILE *stream, const fpos_t *ptr);
```

Positions *stream* at the position recorded by *fgetpos* in *\*ptr*.

**Returns**      zero if successful,  
                 a non-zero value on error.

***ftell***

```
#include <stdio.h>
long ftell(FILE *stream);
```

**Returns**      the current file position for *stream*, or  
                 -1L on error.

***fwrite***

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size,
 size_t nobj, FILE *stream);
```

Writes *nobj* members of *size* bytes to the given *stream* from the array pointed to by *ptr*.

**Returns**      the number of successfully written objects.

***getc***

```
#include <stdio.h>
int getc(FILE *stream);
```

Reads one character out of the given *stream*. Currently #defined as *getchar()*, because FILE I/O is not supported.



See also "*\_read()*".

**Returns**      the character read or EOF on error.

***getchar***

```
#include <stdio.h>
int getchar(void);
```

Reads one character from standard input.



See also ”\_read()”.

**Returns**      the character read or EOF on error.

***getcwd***

```
#include <unistd.h>
char * getcwd(char * buf, size_t size);
```

Use the file system simulation feature of CrossView Pro to retrieve the current directory on the host.

**Returns**      the directory name if successful,  
NULL on error.

***getenv***

```
#include <stdlib.h>
char *getenv(const char *name);
```

**Returns**      the environment string associated with name, or NULL if no string exists.

***gets***

```
#include <stdio.h>
char *gets(char *s);
```

Reads all characters from standard input until a newline is found. The newline is replaced by a NULL-character.



See also ”\_read()”.

**Returns**      a pointer to the read string or NULL on error.

***gmtime***

```
#include <time.h>
struct tm *gmtime(const time_t *tp);
```

Converts the calendar time *\*tp* into Coordinated Universal Time (UTC).

**Returns** a structure representing the UTC, or NULL if UTC is not available.

***isalnum***

```
#include <ctype.h>
int isalnum(int c);
```

**Returns** a non-zero value when *c* is an alphabetic character or a number ([A-Z][a-z][0-9]).

***isalpha***

```
#include <ctype.h>
int isalpha(int c);
```

**Returns** a non-zero value when *c* is an alphabetic character ([A-Z][a-z]).

***isascii***

```
#include <ctype.h>
int isascii(int c);
```

**Returns** a non-zero value when *c* is in the range of 0 and 127. This is a non-ANSI function.

***isctrl***

```
#include <ctype.h>
int isctrl(int c);
```

**Returns** a non-zero value when *c* is a control character.

***isdigit***

```
#include <ctype.h>
int isdigit(int c);
```

**Returns** a non-zero value when *c* is a numeric character ([0-9]).

***isfinite***

```
#include <float.h>
int isfinite(double d);
```

IEEE-754-1985 recommended function. Test the given variable on being a finite (IEEE-754) value.

**Returns** zero if the variable is not finite, else non-zero.

***isgraph***

```
#include <ctype.h>
int isgraph(int c);
```

**Returns** a non-zero value when *c* is printable, but not a space.

***isinf***

```
#include <float.h>
int isinf(double d);
```

IEEE-754-1985 recommended function. Test the given variable on being an infinite (IEEE-754) value.

**Returns** zero if the variable is not +-infinite, else non-zero.

***islower***

```
#include <ctype.h>
int islower(int c);
```

**Returns** a non-zero value when *c* is a lowercase character ([a-z]).

***isnan***

```
#include <float.h>
int isnan(double d);
```

IEEE-754-1985 recommended function. Test the given variable on being a NaN (Not a Number, IEEE-754) value.

**Returns**      zero if the variable is not NaN, else non-zero.

***isprint***

```
#include <ctype.h>
int isprint(int c);
```

**Returns**      a non-zero value when *c* is printable, including spaces.

***ispunct***

```
#include <ctype.h>
int ispunct(int c);
```

**Returns**      a non-zero value when *c* is a punctuation character (such as `','`, `''`, `!`, etc.).

***isspace***

```
#include <ctype.h>
int isspace(int c);
```

**Returns**      a non-zero value when *c* is a space type character (space, tab, vertical tab, formfeed, linefeed, carriage return).

***isupper***

```
#include <ctype.h>
int isupper(int c);
```

**Returns**      a non-zero value when *c* is an uppercase character (`[A-Z]`).



***isxdigit***

```
#include <ctype.h>
int isxdigit(int c);
```

**Returns** a non-zero value when *c* is a hexadecimal digit ([0-9][A-F][a-f]).

***labs***

```
#include <stdlib.h>
long labs(long n);
```

**Returns** the absolute value of the signed long argument.

***ldexp***

```
#include <math.h>
double ldexp(double x, int n);
```

**Returns** the result of:  $x \cdot 2^n$ .

***ldiv***

```
#include <stdlib.h>
ldiv_t ldiv(long num, long denom);
```

Both arguments are long integers. The returned quotient and remainder are also long integers.

**Returns** a structure containing the quotient and remainder of *num* divided by *denom*.

***localeconv***

```
#include <locale.h>
struct lconv *localeconv(void);
```

Sets the components of an object with type `struct lconv` with values appropriate for the formatting of numeric quantities according to the rules of the current locale.

**Returns** a pointer to the filled-in object.

***localtime***

```
#include <time.h>
struct tm *localtime(const time_t *tp);
```

Converts the calendar time `*tp` into local time.

**Returns** a structure representing the local time.

***log***

```
#include <math.h>
double log(double x);
```

**Returns** the natural logarithm  $\ln(x)$ ,  $x > 0$ .

***log10***

```
#include <math.h>
double log10(double x);
```

**Returns** the base 10 logarithm  $\log_{10}(x)$ ,  $x > 0$ .

***longjmp***

```
#include <setjmp.h>
void longjmp(jmp_buf env, int val);
```

Restores the environment previously saved with a call to `setjmp()`. The function calling the corresponding call to `setjmp()` may not be terminated yet. The value of `val` may not be zero.

**Returns**      nothing.

***lseek***

```
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
```

Moves read-write file offset. This function calls `_lseek`.

**Returns**      the resulting pointer location if successful,  
                 -1 on error.

***malloc***

```
#include <stdlib.h>
void *malloc(size_t size);
```

The allocated space is not initialized. The maximum space that can be allocated can be changed by customizing the heap size (see the section *Heap*). By default no heap is allocated. When "malloc()" is used while no heap is defined, the locator gives an error.

**Returns**      a pointer to space in external memory of `size` bytes length.  
                 NULL if there is not enough space left.

***mblen***

```
#include <stdlib.h>
int mblen(const char *s, size_t n);
```

Determines the number of bytes comprising the multi-byte character pointed to by *s*, if *s* is not a null pointer. Except that the shift state is not affected. At most *n* characters will be examined, starting at the character pointed to by *s*.

**Returns**      the number of bytes, or 0 if *s* points to the null character, or -1 if the bytes do not form a valid multi-byte character.

***mbstowcs***

```
#include <stdlib.h>
size_t mbstowcs(wchar_t *pwcs,
 const char *s, size_t n);
```

Converts a sequence of multi-byte characters that begins in the initial shift state from the array pointed to by *s*, into a sequence of corresponding codes and stores these codes into the array pointed to by *pwcs*, stopping after *n* codes are stored or a code with value zero is stored.

**Returns**      the number of array elements modified (not including a terminating zero code, if any), or (size\_t)-1 if an invalid multi-byte character is encountered.

***mbtowc***

```
#include <stdlib.h>
int mbtowc(wchar_t *pwc,
 const char *s, size_t n);
```

Determines the number of bytes that comprise the multi-byte character pointed to by *s*. It then determines the code for value of type *wchar\_t* that corresponds to that multi-byte character. If the multi-byte character is valid and *pwc* is not a null pointer, the *mbtowc* function stores the code in the object pointed to by *pwc*. At most *n* characters will be examined, starting at the character pointed to by *s*.

**Returns**      the number of bytes, or 0 if *s* points to the null character, or -1 if the bytes do not form a valid multi-byte character.

***memchr***

```
#include <string.h>
void *memchr(const void *cs, int c, size_t n);
```

Checks the first *n* bytes of *cs* on the occurrence of character *c*.

**Returns**      NULL when not found, otherwise a pointer to the found character is returned.

***memcmp***

```
#include <string.h>
int memcmp(const void *cs, const void *ct,
 size_t n);
```

Compares the first *n* bytes of *cs* with the contents of *ct*.

**Returns**      a value < 0 if *cs* < *ct*,  
                 0 if *cs* == *ct*,  
                 or a value > 0 if *cs* > *ct*.

***memcpy***

```
#include <string.h>
void *memcpy(void *s, const void *ct, size_t n);
```

Copies *n* characters from *ct* to *s*. No care is taken if the two objects overlap.

**Returns**      *s*

***memmove***

```
#include <string.h>
void *memmove(void *s, const void *ct, size_t n);
```

Copies *n* characters from *ct* to *s*. Overlapping objects will be handled correctly.

**Returns**      *s*

***memset***

```
#include <string.h>
void *memset(void *s, int c, size_t n);
```

Fills the first *n* bytes of *s* with character *c*.

**Returns**        *s*

***mktime***

```
#include <time.h>
time_t mktime(struct tm *tp);
```

Converts the local time in the structure *\*tp* into calendar time.

**Returns**        the calendar time, or -1 if it cannot be represented.

***modf***

```
#include <math.h>
double modf(double x, double *ip);
```

Splits *x* into integral and fractional parts, each with the same sign as *x*. It stores the integral part in *\*ip*.

**Returns**        the fractional part.

***offsetof***

```
#include <stddef.h>
int offsetof(type, member);
```

**Returns**        the offset for the given member in an object of type.

***open***

```
#include <fcntl.h>
int open(const char * name, int flags);
```

Opens a file a file for reading or writing. This function calls `_open`.



See also "fopen()".

**Returns** the file descriptor if successful (a non-negative integer), or -1 on error.

***perror***

```
#include <stdio.h>
void perror(const char *s);
```

Prints `s` and an implementation-defined error message corresponding to the integer `errno`, as if by:

```
fprintf(stderr, "%s: %s\n", s, "error message");
```

The contents of the error message are the same as those returned by the `strerror` function with the argument `errno`.



See also the "strerror()" function.

**Returns** nothing.

***pow***

```
#include <math.h>
double pow(double x, double y);
```

A domain error occurs if  $x=0$  and  $y \leq 0$ , or if  $x < 0$  and  $y$  is not an integer.

**Returns** the result of  $x$  raised to the power of  $y$ :  $x^y$ .

***printf***

```
#include <stdio.h>
int printf(const char *format,...);
```

Performs a formatted write to the standard output stream.



See also `__write()`.

**Returns** the number of characters written to the output stream.

The `format` string may contain plain text mixed with conversion specifiers. Each conversion specifier should be preceded by a `'%` character. The conversion specifier should be build in order:

- Flags (in any order):
  - specifies left adjustment of the converted argument.
  - + a number is always preceded with a sign character.  
+ has higher precedence as space.
  - space a negative number is preceded with a sign, positive numbers with a space.
  - 0 specifies padding to the field width with zeros (only for numbers).
  - # specifies an alternate output form. For `o`, the first digit will be zero. For `x` or `X`, `"0x"` and `"0X"` will be prefixed to the number. For `e`, `E`, `f`, `g`, `G`, the output always contains a decimal point, trailing zeros are not removed.
- A number specifying a minimum field width. The converted argument is printed in a field with at least the length specified here. If the converted argument has fewer characters than specified, it will be padded at the left side (or at the right when the flag `'-'` was specified) with spaces. Padding to numeric fields will be done with zeros when the flag `'0'` is also specified (only when padding left). Instead of a numeric value, also `'*'` may be specified, the value is then taken from the next argument, which is assumed to be of type `int`.
- A period. This separates the minimum field width from the precision.



- A number specifying the maximum length of a string to be printed. Or the number of digits printed after the decimal point (only for floating point conversions). Or the minimum number of digits to be printed for an integer conversion. Instead of a numeric value, also '\*' may be specified, the value is then taken from the next argument, which is assumed to be of type int.
- A length modifier 'h', 'l' or 'L'. 'h' indicates that the argument is to be treated as a short or unsigned short number. 'l' should be used if the argument is a long integer. 'L' indicates that the argument is a long double.

Flags, length specifier, period, precision and length modifier are optional, the conversion character is not. The conversion character must be one of the following, if a character following '%' is not in the list, the behavior is undefined:

| Character | Printed as                                                                                                                                                  |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| d, i      | int, signed decimal                                                                                                                                         |
| o         | int, unsigned octal                                                                                                                                         |
| x, X      | int, unsigned hexadecimal in lowercase or uppercase respectively                                                                                            |
| u         | int, unsigned decimal                                                                                                                                       |
| c         | int, single character (converted to unsigned char)                                                                                                          |
| s         | char *, the characters from the string are printed until a NULL character is found. When the given precision is met before, printing will also stop         |
| f         | double                                                                                                                                                      |
| e, E      | double                                                                                                                                                      |
| g, G      | double                                                                                                                                                      |
| n         | int *, the number of characters written so far is written into the argument. This should be a pointer to an integer in default memory. No value is printed. |
| p         | pointer (hexadecimal 24-bit value)                                                                                                                          |
| r         | _fract, _sfract                                                                                                                                             |
| R         | _accum                                                                                                                                                      |
| %         | No argument is converted, a '%' is printed.                                                                                                                 |

Table 6-2: Printf conversion characters

***putc***

```
#include <stdio.h>
int putc(int c, FILE *stream);
```

Puts one character onto the given stream.



See also "\_write()".

**Returns** EOF on error.

***putchar***

```
#include <stdio.h>
int putchar(int c);
```

Puts one character onto standard output.



See also "\_write()".

**Returns** the character written or EOF on error.

***puts***

```
#include <stdio.h>
int puts(const char *s);
```

Writes the string to stdout, the string is terminated by a newline.



See also "\_write()".

**Returns** NULL if successful, or EOF on error.

***qsort***

```
#include <stdlib.h>
void qsort(
 const void *base, size_t n, size_t size,
 int (* cmp)(const void *, const void *));
```

This function sorts an array of *n* members. The initial base of the array is given by *base*. The size of each member is specified by *size*. The given array is sorted in ascending order, according to the results of the function pointed to by *cmp*.

**Returns**      nothing.

***raise***

```
#include <signal.h>
int raise(int sig);
```

Sends the signal *sig* to the program.



See also "signal()".

**Returns**      zero if successful, or a non-zero value if unsuccessful.

***rand***

```
#include <stdlib.h>
int rand(void);
```

**Returns**      a sequence of pseudo-random integers, in the range 0 to RAND\_MAX.

***read***

```
#include <unistd.h>
size_t read(int fd, char * buffer, size_t count);
```

Reads a sequence of characters from a file. This function calls `_read`.



See also "`_read()`".

***realloc***

```
#include <stdlib.h>
void *realloc(void *p, size_t size);
```

Reallocates the space for the object pointed to by *p*. The contents of the object will be the same as before calling `realloc()`. The maximum space that can be allocated can be changed by customizing the heap size (see the section *Heap*). By default no heap is allocated. When "realloc()" is used while no heap is defined, the linker gives an error.



See also "malloc".

**Returns**      NULL and *\*p* is not changed, if there is not enough space for the new allocation. Otherwise a pointer to the newly allocated space for the object is returned.

***remove***

```
#include <stdio.h>
int remove(const char *filename);
```

Removes the named file, so that a subsequent attempt to open it fails.

**Returns**      zero if file is successfully removed, or  
a non-zero value, if the attempt fails.

***rename***

```
#include <stdio.h>
int rename(const char *oldname,
 const char *newname);
```

Changes the name of the file.

**Returns**      zero if file is successfully renamed, or  
a non-zero value, if the attempt fails.

***rewind***

```
#include <stdio.h>
void rewind(FILE *stream);
```

Sets the file position indicator for the stream pointed to by `stream` to the beginning of the file. This function is equivalent to:

```
(void) fseek(stream, 0L, SEEK_SET);
clearerr(stream);
```

**Returns**      nothing.

***scalb***

```
#include <float.h>
double scalb(double d, int power);
```

IEEE-754-1985 Recommended function.

**Returns**       $d * 2^{\text{power}}$  for integral values power without computing  $2^N$ .

***scanf***

```
#include <stdio.h>
int scanf(const char *format, ...);
```

Performs a formatted read from the standard input stream.



See also "`_read()`".

**Returns**      the number of items converted successfully.

All arguments to this function should be pointers to variables (in default memory) of the type which is specified in the format string.

The format string may contain :

- Blanks or tabs, which are skipped.
- Normal characters (not '%'), which should be matched exactly in the input stream.
- Conversion specifications, starting with a '%' character.

Conversion specifications should be built as follows (in order) :

- A `*`, meaning that no assignment is done for this field.
- A number specifying the maximum field width.
- The conversion characters `d`, `i`, `n`, `o`, `u` and `x` can be preceded by `'h'` if the argument is a pointer to `short` rather than `int`, or by `'l'` (letter ell) if the argument is a pointer to `long`. The conversion characters `e`, `f`, and `g` can be preceded by `'l'` if a pointer `double` rather than `float` is in the argument list, and by `'L'` if a pointer to a `long double`.
- A conversion specifier. `*`, maximum field width and length modifier are optional, the conversion character is not. The conversion character must be one of the following, if a character following `'%'` is not in the list, the behavior is undefined.

Length specifier and length modifier are optional, the conversion character is not. The conversion character must be one of the following, if a character following `'%'` is not in the list, the behavior is undefined.

| Character                       | Scanned as                                                                                                                                                                           |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>d</code>                  | <code>int</code> , signed decimal.                                                                                                                                                   |
| <code>i</code>                  | <code>int</code> , the integer can be octal (i.e. with a leading 0) or hexadecimal (leading <code>"0x"</code> or <code>"0X"</code> ), or just decimal.                               |
| <code>o</code>                  | <code>int</code> , unsigned octal.                                                                                                                                                   |
| <code>u</code>                  | <code>int</code> , unsigned decimal.                                                                                                                                                 |
| <code>x</code>                  | <code>int</code> , unsigned hexadecimal in lowercase or uppercase.                                                                                                                   |
| <code>c</code>                  | single character (converted to unsigned char).                                                                                                                                       |
| <code>s</code>                  | <code>char *</code> , a string of non white space characters. The argument should point to an array of characters, large enough to hold the string and a terminating NULL character. |
| <code>f</code>                  | <code>float</code>                                                                                                                                                                   |
| <code>e</code> , <code>E</code> | <code>float</code>                                                                                                                                                                   |
| <code>g</code> , <code>G</code> | <code>float</code>                                                                                                                                                                   |
| <code>n</code>                  | <code>int *</code> , the number of characters written so far is written into the argument. No scanning is done.                                                                      |
| <code>p</code>                  | pointer; hexadecimal 24-bit value which must be entered without <code>0x-</code> prefix.                                                                                             |
| <code>r</code>                  | <code>_fract</code> , <code>_sfract</code>                                                                                                                                           |

| Character | Scanned as                                                                                                                                                                                                |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| R         | _accum                                                                                                                                                                                                    |
| [...]     | Matches a string of input characters from the set between the brackets. A NULL character is added to terminate the string. Specifying [...] includes the ']' character in the set of scanning characters. |
| [^...]    | Matches a string of input characters not in the set between the brackets. A NULL character is added to terminate the string. Specifying [^...] includes the ']' character in the set.                     |
| %         | Literal '%', no assignment is done.                                                                                                                                                                       |

Table 6-3: *Scanf conversion characters*

**setbuf**

```
#include <stdio.h>
void setbuf(FILE *stream, char *buf);
```

Buffering is turned off for the `stream`, if `buf` is `NULL`. Otherwise, `setbuf` is equivalent to:

```
(void) setvbuf(stream, buf, _IOFBF, BUFSIZ)
```

**Returns**      nothing.



See also "setvbuf()".

**setjmp**

```
#include <setjmp.h>
int setjmp(jmp_buf env);
```

Saves the current environment for a subsequent call to `longjmp`.

**Returns**      0 after a direct call to `setjmp()`. Calling the function "longjmp()" using the saved `env` restores the current environment and jumps to this place with a non-zero return value.



See also "longjmp()".

***setlocale***

```
#include <locale.h>
char *setlocale(int category, const char *locale);
```

Selects the appropriate portion of the program's locale as specified by the `category` and `locale` arguments.

**Returns**      the string associated with the specified `category` for the new locale if the selection can be honored.  
                 null pointer if the selection cannot be honored.

***setvbuf***

```
#include <stdio.h>
int setvbuf(FILE *stream, char *buf,
 int mode, size_t size);
```

Controls buffering for the `stream`; this function must be called before reading or writing. `mode` can have the following values:

`_IOFBF` causes full buffering  
`_IOLBF` causes line buffering of text files  
`_IONBF` causes no buffering

If `buf` is not NULL, it will be used as a buffer; otherwise a buffer will be allocated. `size` determines the buffer size.

**Returns**      zero if successful  
                 a non-zero value for an error.



See also "setbuf()".



***signal***

```
#include <signal.h>
void (*signal(int sig, void (*handler)(int)))(int);
```

Determines how subsequent signals will be handled. If `handler` is `SIG_DFL`, the default behavior is used; if `handler` is `SIG_IGN`, the signal is ignored; otherwise, the function pointed to by `handler` will be called, with the argument of the type of signal. Valid signals are:

|                      |                                                           |
|----------------------|-----------------------------------------------------------|
| <code>SIGABRT</code> | abnormal termination, e.g. from <code>abort</code>        |
| <code>SIGFPE</code>  | arithmetic error, e.g. zero divide or overflow            |
| <code>SIGILL</code>  | illegal function image, e.g. illegal instruction          |
| <code>SIGINT</code>  | interactive attention, e.g. interrupt                     |
| <code>SIGSEGV</code> | illegal storage access, e.g. access outside memory limits |
| <code>SIGTERM</code> | termination request sent to this program                  |

When a signal `sig` subsequently occurs, the signal is restored to its default behavior; then the signal-handler function is called, as if by `(*handler)(sig)`. If the handler returns, the execution will resume where it was when the signal occurred.

**Returns** the previous value of `handler` for the specific signal, or `SIG_ERR` if an error occurs.

***sin***

```
#include <math.h>
double sin(double x);
```

**Returns** the sine of `x`.

***sinh***

```
#include <math.h>
double sinh(double x);
```

**Returns** the hyperbolic sine of `x`.

***sprintf***

```
#include <stdio.h>
int sprintf(char *s, const char *format, ...);
```

Performs a formatted write to a string.



See also "printf".

***sqrt***

```
#include <math.h>
double sqrt(double x);
```

**Returns** the square root of  $x$ .  $\sqrt{x}$ , where  $x \geq 0$ .

***srand***

```
#include <stdlib.h>
void srand(unsigned int seed);
```

This function uses *seed* as the start of a new sequence of pseudo-random numbers to be returned by subsequent calls to `srand()`. When `srand` is called with the same seed value, the sequence of pseudo-random numbers generated by `rand()` will be repeated.

**Returns** pseudo random numbers.

***sscanf***

```
#include <stdio.h>
int sscanf(char *s, const char *format, ...);
```

Performs a formatted read from a string.



See also "scanf".

***stat***

```
#include <unistd.h>
int stat(const char * name, struct stat * buf);
```

Use the file system simulation feature of CrossView Pro to stat() a file on the host platform.

**Returns**      zero if successful,  
                 -1 on error.

***strcat***

```
#include <string.h>
char *strcat(char *s, const char *ct);
```

Concatenates string ct to string s, including the trailing NULL character.

**Returns**      s

***strchr***

```
#include <string.h>
char *strchr(const char *cs, int c);
```

**Returns**      a pointer to the first occurrence of character c in the string cs. If not found, NULL is returned.

***strcmp***

```
#include <string.h>
int strcmp(const char *cs, const char *ct);
```

Compares string cs to string ct.

**Returns**      <0 if cs < ct,  
                 0 if cs == ct,  
                 >0 if cs > ct.

***strcoll***

```
#include <string.h>
int strcoll(const char *cs, const char *ct);
```

Compares string *cs* to string *ct*. The comparison is based on strings interpreted as appropriate to the program's locale.

**Returns**      <0 if *cs* < *ct*,  
                 0 if *cs* = *ct*,  
                 >0 if *cs* > *ct*.

***strcpy***

```
#include <string.h>
char *strcpy(char *s, const char *ct);
```

Copies string *ct* into the string *s*, including the trailing NULL character.

**Returns**      *s*

***strcspn***

```
#include <string.h>
size_t strcspn(const char *cs, const char *ct);
```

**Returns**      the length of the prefix in string *cs*, consisting of characters not in the string *ct*.

***strerror***

```
#include <string.h>
char *strerror(size_t n);
```

**Returns**      pointer to implementation-defined string corresponding to error *n*.

***strftime***

```
#include <time.h>
size_t strftime(char *s, size_t smax,
 const char *fmt,
 const struct tm *tp);
```

Formats date and time information from the structure *\*tp* into *s* according to the specified format *fmt*. *fmt* is analogous to a *printf* format. Each *%c* is replaced as described below:

|           |                                                                  |
|-----------|------------------------------------------------------------------|
| <i>%a</i> | abbreviated weekday name                                         |
| <i>%A</i> | full weekday name                                                |
| <i>%b</i> | abbreviated month name                                           |
| <i>%B</i> | full month name                                                  |
| <i>%c</i> | local date and time representation                               |
| <i>%d</i> | day of the month (01–31)                                         |
| <i>%H</i> | hour, 24-hour clock (00–23)                                      |
| <i>%I</i> | hour, 12-hour clock (01–12)                                      |
| <i>%j</i> | day of the year (001–366)                                        |
| <i>%m</i> | month (01–12)                                                    |
| <i>%M</i> | minute (00–59)                                                   |
| <i>%p</i> | local equivalent of AM or PM                                     |
| <i>%S</i> | second (00–59)                                                   |
| <i>%U</i> | week number of the year, Sunday as first day of the week (00–53) |
| <i>%w</i> | weekday (0–6, Sunday is 0)                                       |
| <i>%W</i> | week number of the year, Monday as first day of the week (00–53) |
| <i>%x</i> | local date representation                                        |
| <i>%X</i> | local time representation                                        |
| <i>%y</i> | year without century (00–99)                                     |
| <i>%Y</i> | year with century                                                |
| <i>%Z</i> | time zone name, if any                                           |
| <i>%%</i> | <i>%</i>                                                         |

Ordinary characters (including the terminating ‘\0’) are copied into *s*. No more than *smax* characters are placed into *s*.

**Returns** the number of characters (‘\0’ not included), or zero if more than *smax* characters were produced.

***strlen***

```
#include <string.h>
size_t strlen(const char *cs);
```

**Returns** the length of the string in *cs*, not counting the NULL character.

***strncat***

```
#include <string.h>
char *strncat(char *s, const char *ct, size_t n);
```

Concatenates string *ct* to string *s*, at most *n* characters are copied. Add a trailing NULL character.

**Returns** *s*

***strncmp***

```
#include <string.h>
int strncmp(const char *cs, const char *ct,
 size_t n);
```

Compares at most *n* bytes of string *cs* to string *ct*.

**Returns** <0 if *cs* < *ct*,  
0 if *cs* == *ct*,  
>0 if *cs* > *ct*.

***strncpy***

```
#include <string.h>
char *strncpy(char *s, const char *ct, size_t n);
```

Copies string *ct* onto the string *s*, at most *n* characters are copied. Add a trailing NULL character if the string is smaller than *n* characters.

**Returns** *s*

***strpbrk***

```
#include <string.h>
char *strpbrk(const char *cs, const char *ct);
```

**Returns** a pointer to the first occurrence in *cs* of any character out of string *ct*. If none are found, NULL is returned.

***strrchr***

```
#include <string.h>
char *strrchr(const char *cs, int c);
```

**Returns** a pointer to the last occurrence of *c* in the string *cs*. If not found, NULL is returned.

***strspn***

```
#include <string.h>
size_t strspn(const char *cs, const char *ct);
```

**Returns** the length of the prefix in string *cs*, consisting of characters in the string *ct*.

***strstr***

```
#include <string.h>
char *strstr(const char *cs, const char *ct);
```

**Returns** a pointer to the first occurrence of string *ct* in the string *cs*. Returns NULL if not found.

***strtod***

```
#include <stdlib.h>
double strtod(const char *s, char **endp);
```

Converts the initial portion of the string pointed to by *s* to a double value. Initial white spaces are skipped. When *endp* is not a NULL pointer, after this function is called, *\*endp* will point to the first character not used by the conversion.

**Returns**      the read value.

***strtok***

```
#include <string.h>
char *strtok(char *s, const char *ct);
```

Search the string *s* for tokens delimited by characters from string *ct*. It terminates the token with a NULL character.

**Returns**      a pointer to the token. A subsequent call with *s* == NULL will return the next token in the string.

***strtol***

```
#include <stdlib.h>
long strtol(const char *s, char **endp, int base);
```

Converts the initial portion of the string pointed to by *s* to a long integer. Initial white spaces are skipped. Then a value is read using the given *base*. When *base* is zero, the *base* is taken as defined for integer constants. I.e. numbers starting with an '0' are taken octal, numbers starting with '0x' or '0X' are taken hexadecimal. Other numbers are taken decimal. When *endp* is not a NULL pointer, after this function is called, *\*endp* will point to the first character not used by the conversion.

**Returns**      the read value.



***strtoul***

```
#include <stdlib.h>
unsigned long strtoul(
 const char *s, char **endp, int base);
```

Converts the initial portion of the string pointed to by *s* to an unsigned long integer. Initial white spaces are skipped. Then a value is read using the given base. When *base* is zero, the base is taken as defined for integer constants. I.e. numbers starting with an '0' are taken octal, numbers starting with '0x' or '0X' are taken hexadecimal. Other numbers are taken decimal. When *endp* is not a NULL pointer, after this function is called, *\*endp* will point to the first character not used by the conversion.

**Returns**      the read value.

***strxfrm***

```
#include <string.h>
size_t
strncmp(char *ct, const char *cs, size_t n);
```

Transforms the string pointed to by *cs* and places the resulting string into the array pointed to by *ct*. No more than *n* characters are placed into the resulting string pointed to by *ct*, including the terminating null character.

**Returns**      the length of the transformed string.

***system***

```
#include <stdlib.h>
int system(const char *s);
```

Passes the string *s* to the environment for execution.

**Returns**      a non-zero value if there is a command processor, if *s* is NULL; or an implementation-dependent value, if *s* is not NULL.

***tan***

```
#include <math.h>
double tan(double x);
```

**Returns** the tangent of x.

***tanh***

```
#include <math.h>
double tanh(double x);
```

**Returns** the hyperbolic tangent of x.

***time***

```
#include <time.h>
time_t time(time_t *tp);
```

The return value is also assigned to \*tp, if tp is not NULL.

**Returns** the current calendar time, or -1 if the time is not available.

***tmpfile***

```
#include <stdio.h>
FILE *tmpfile(void);
```

Creates a temporary file of the mode "wb+" that will be automatically removed when closed or when the program terminates normally.

**Returns** a stream if successful, or NULL if the file could not be created.

***tmpnam***

```
#include <stdio.h>
char *tmpnam(char s[L_tmpnam]);
```

Creates a temporary name (not a file). Each time `tmpnam` is called a different name is created.

`tmpnam(NULL)` creates a string that is not the name of an existing file, and returns a pointer to an internal static array. `tmpnam(s)` creates a string and stores it in `s` and also returns it as the function value. `s` must have room for at least `L_tmpnam` characters. At most `TMP_MAX` different names are guaranteed during execution of the program.

**Returns** a pointer to the temporary name, as described above.

***toascii***

```
#include <ctype.h>
int toascii(int c);
```

Converts `c` to an ascii value (strip highest bit). This is a non-ANSI function.

**Returns** the converted value.

***tolower***

```
#include <ctype.h>
int tolower(int c);
```

**Returns** `c` converted to a lowercase character if it is an uppercase character, otherwise `c` is returned.

***toupper***

```
#include <ctype.h>
int toupper(int c);
```

**Returns** `c` converted to an uppercase character if it is a lowercase character, otherwise `c` is returned.

***ungetc***

```
#include <stdio.h>
int ungetc(int c, FILE *fin);
```

Pushes at the most one character back onto the input buffer.

**Returns** EOF on error.

***unlink***

```
#include <unistd.h>
int unlink(const char * name);
```

Removes the named file, so that a subsequent attempt to open it fails. This function calls `_unlink`.

**Returns** zero if file is successfully removed, or a non-zero value, if the attempt fails.

***va\_arg***

```
#include <stdarg.h>
va_arg(va_list ap, type);
```

**Returns** the value of the next argument in the variable argument list. It's return type has the type of the given argument type. A next call to this macro will return the value of the next argument.

***va\_end***

```
#include <stdarg.h>
va_end(va_list ap);
```

This macro must be called after the arguments have been processed. It should be called before the function using the macro 'va\_start' is terminated (ANSI specification).

***va\_start***

```
#include <stdarg.h>
va_start(va_list ap, lastarg);
```

This macro initializes `ap`. After this call, each call to `va_arg()` will return the value of the next argument. In our implementation, `va_list` cannot contain any bit type variables. Also the given argument `lastarg` must be the last non-bit type argument in the list.

***vfprintf***

```
#include <stdio.h>
int vfprintf(FILE *stream,
 const char *format, va_list arg);
```

Is equivalent to `vprintf`, but writes to the given stream.



See also "`vprintf()`", "`_write()`".

***vprintf***

```
#include <stdio.h>
int vprintf(const char *format, va_list arg);
```

Does a formatted write to standard output. Instead of a variable argument list as for `printf()`, this function expects a pointer to the list.



See also "`printf()`", "`_write()`".

***vsprintf***

```
#include <stdio.h>
int vsprintf(char *s, const char *format,
 va_list arg);
```

Does a formatted write a string. Instead of a variable argument list as for `printf()`, this function expects a pointer to the list.



See also "`printf()`", "`_write()`".

***wcstombs***

```
#include <stdlib.h>
size_t wcstombs(char *s, const wchar_t *pwcs,
 size_t n);
```

Converts a sequence of codes that correspond to multi-byte characters from the array pointed to by `pwcs`, into a sequence of multi-byte characters that begins in the initial shift state and stores these multi-byte characters into the array pointed to by `s`, stopping if a multi-byte character would exceed the limit of `n` total bytes or if a null character is stored.

**Returns** the number of bytes modified (not including a terminating null character, if any), or `(size_t)-1` if a code is encountered that does not correspond to a valid multi-byte character.

***wctomb***

```
#include <stdlib.h>
int wctomb(char *s, wchar_t wchar);
```

Determines the number of bytes needed to represent the multi-byte corresponding to the code whose value is `wchar` (including any change in the shift state). It stores the multi-byte character representation in the array pointed to by `s` (if `s` is not a null pointer). At most `MB_CUR_MAX` characters are stored. If the value of `wchar` is zero, the `wctomb` function is left in the initial shift state.

**Returns** the number of bytes, or `-1` if the value of `wchar` does not correspond to a valid multi-byte character.

***write***

```
#include <unistd.h>
size_t write(int fd, char * buffer, size_t count);
```

Write a sequence of characters to a file. This function calls `_write`.



See also `"_write()"`.

6.3.4 C LIBRARY REENTRANCY

Some of the functions in the C library are reentrant, others are not. The table below shows the functions in the C library, whether they are reentrant and, if not, the reason why. Note that some of the functions are not reentrant because they set the global variable 'errno'. If your program does not check this variable and errno is the only reason for the function not being reentrant, these functions can be assumed reentrant as well.

The explanation of the cause why a function is not reentrant sometimes refers to a footnote because the explanation is too lengthy for the table.

| Function | Reentrant | Cause                                                                             |
|----------|-----------|-----------------------------------------------------------------------------------|
| abort    | no        | Calls exit                                                                        |
| abs      | yes       | —                                                                                 |
| access   | no        | Uses global File System Simulation buffer, _fss_buffer                            |
| acos     | no        | Function sets errno when error occurs. If errno not used, acos is reentrant.      |
| asctime  | no        | asctime defines static array for broken-down time string.                         |
| asin     | no        | Function sets errno when error occurs. If errno not used, asin is reentrant.      |
| atan     | yes       | —                                                                                 |
| atan2    | yes       | —                                                                                 |
| atexit   | no        | atexit defines static array with function pointers to execute at exit of program. |
| atof     | yes       | —                                                                                 |
| atoi     | yes       | —                                                                                 |
| atol     | yes       | —                                                                                 |
| bsearch  | yes       | —                                                                                 |
| calloc   | no        | calloc uses static buffer management structures. See malloc (5).                  |
| ceil     | yes       | —                                                                                 |
| chdir    | no        | Uses global File System Simulation buffer, _fss_buffer                            |
| cleanup  | no        | Calls fclose. See (1)                                                             |
| clearerr | no        | Modifies iob[]. See (1)                                                           |

| Function              | Reentrant | Cause                                                                                                                                                                         |
|-----------------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| clock                 | yes       | —                                                                                                                                                                             |
| close                 | no        | Calls <code>_close</code>                                                                                                                                                     |
| <code>_close</code>   | no        | Uses global File System Simulation buffer, <code>_fss_buffer</code>                                                                                                           |
| cos                   | yes       | —                                                                                                                                                                             |
| cosh                  | no        | cosh calls <code>exp()</code> , which sets <code>errno</code> . If <code>errno</code> is discarded, cosh is reentrant.                                                        |
| ctime                 | no        | Calls <code>asctime</code>                                                                                                                                                    |
| difftime              | yes       | —                                                                                                                                                                             |
| div                   | yes       | —                                                                                                                                                                             |
| <code>_doflt</code>   | no        | Uses I/O functions which modify <code>iob[]</code> . See (1).                                                                                                                 |
| <code>_doprint</code> | no        | Uses indirect access to static <code>iob[]</code> array. See (1).                                                                                                             |
| <code>_doscan</code>  | no        | Uses indirect access to <code>iob[]</code> and calls <code>ungetc</code> (access to local static <code>ungetc[]</code> buffer). See (1).                                      |
| exit                  | no        | Calls <code>fclose</code> indirectly which uses <code>iob[]</code> calls functions in <code>_atexit</code> array. See (1). To make exit reentrant kernel support is required. |
| exp                   | no        | Sets <code>errno</code> . If <code>errno</code> not used, exp is reentrant.                                                                                                   |
| fabs                  | yes       | —                                                                                                                                                                             |
| fclose                | no        | Uses values in <code>iob[]</code> . See (1).                                                                                                                                  |
| feof                  | no        | Uses values in <code>iob[]</code> . See (1).                                                                                                                                  |
| ferror                | no        | Uses values in <code>iob[]</code> . See (1).                                                                                                                                  |
| fflush                | no        | Modifies <code>iob[]</code> . See (1).                                                                                                                                        |
| fgetc                 | no        | Uses pointer to <code>iob[]</code> . See (1).                                                                                                                                 |
| fgetpos               | no        | Sets the variable <code>errno</code> and uses pointer to <code>iob[]</code> . See (1) / (2).                                                                                  |
| fgets                 | no        | Uses <code>iob[]</code> . See (1).                                                                                                                                            |
| <code>_filbuf</code>  | no        | Uses <code>iob[]</code> . See (1).                                                                                                                                            |
| floor                 | yes       | —                                                                                                                                                                             |
| <code>_flsbuf</code>  | no        | Uses <code>iob[]</code> . See (1).                                                                                                                                            |
| fmod                  | yes       | —                                                                                                                                                                             |



| Function | Reentrant | Cause                                                               |
|----------|-----------|---------------------------------------------------------------------|
| fopen    | no        | Uses iob[] and calls malloc when file open for buffered IO. See (1) |
| fprintf  | no        | Uses iob[]. See (1).                                                |
| fputc    | no        | Uses iob[]. See (1).                                                |
| fputs    | no        | Uses iob[]. See (1).                                                |
| fread    | no        | Calls fgetc. See (1).                                               |
| free     | no        | free uses static buffer management structures. See malloc (5).      |
| freopen  | no        | Modifies iob[]. See (1).                                            |
| frexp    | yes       | —                                                                   |
| fscanf   | no        | Uses iob[]. See (1)                                                 |
| fseek    | no        | Uses iob[] and calls _doscan. Accesses ungetc[] array. See (1).     |
| fsetpos  | no        | Uses iob[] and sets errno. See (1) / (2).                           |
| ftell    | no        | Uses iob[] and sets errno. Calls _lseek. See (1) / (2).             |
| fwrite   | no        | Uses iob[]. See (1).                                                |
| getc     | no        | Uses iob[]. See (1).                                                |
| getchar  | no        | Uses iob[]. See (1).                                                |
| getcwd   | no        | Uses global File System Simulation buffer, _fss_buffer              |
| getenv   | yes       | Skeleton only.                                                      |
| _getflt  | no        | Uses iob[ ]. See (1).                                               |
| gets     | no        | Uses iob[ ]. See (1).                                               |
| gmtime   | no        | gmtime defines static structure                                     |
| halloc   | no        | Needs kernel support. See malloc (5).                               |
| halloc   | no        | halloc uses static buffer management structures. See malloc (5).    |
| hfree    | no        | hfree uses static buffer management structures. See malloc (5).     |
| hrealloc | no        | See malloc (5).                                                     |
| _iob     | no        | Defines static iob[]. See (1).                                      |
| _ioread  | no        | Depends on low level I/O implementation. Uses iob[]. See (1).       |

| Function   | Reentrant | Cause                                                          |
|------------|-----------|----------------------------------------------------------------|
| _iowrite   | no        | Depends on low level I/O implementation. Uses iob[ ]. See (1). |
| isalnum    | yes       | —                                                              |
| isalpha    | yes       | —                                                              |
| isascii    | yes       | —                                                              |
| iscntrl    | yes       | —                                                              |
| isdigit    | yes       | —                                                              |
| isgraph    | yes       | —                                                              |
| islower    | yes       | —                                                              |
| isprint    | yes       | —                                                              |
| ispunct    | yes       | —                                                              |
| isspace    | yes       | —                                                              |
| isupper    | yes       | —                                                              |
| isxdigit   | yes       | —                                                              |
| _itoa      | yes       | —                                                              |
| labs       | yes       | —                                                              |
| ldexp      | no        | Sets errno. See (2).                                           |
| ldiv       | yes       | —                                                              |
| localeconv | —         | Skeleton function                                              |
| localtime  | yes       | —                                                              |
| log        | no        | Sets errno. See (2).                                           |
| log10      | no        | Calls log. See (2).                                            |
| longjmp    | yes       | —                                                              |
| lseek      | no        | Calls _lseek                                                   |
| _lseek     | no        | Uses global File System Simulation buffer, _fss_buffer         |
| ltoa       | yes       | —                                                              |
| malloc     | no        | Needs kernel support. See (5).                                 |
| mblen      | —         | Skeleton function                                              |
| mbstowcs   | —         | Skeleton function                                              |
| mbtowc     | —         | Skeleton function                                              |
| memchr     | yes       | —                                                              |

| Function           | Reentrant | Cause                                                                                                                                   |
|--------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------|
| memcmp             | yes       | —                                                                                                                                       |
| memcpy             | yes       | —                                                                                                                                       |
| memmove            | yes       | —                                                                                                                                       |
| memset             | yes       | —                                                                                                                                       |
| mktime             | yes       | —                                                                                                                                       |
| modf               | yes       | —                                                                                                                                       |
| open               | no        | Calls <code>_open</code>                                                                                                                |
| <code>_open</code> | no        | Uses global File System Simulation buffer, <code>_fss_buffer</code>                                                                     |
| perror             | no        | Uses <code>errno</code> . See (2)                                                                                                       |
| pow                | no        | Sets <code>errno</code> . See (2)                                                                                                       |
| printf             | no        | Uses <code>io[ ]</code> . See (1)                                                                                                       |
| putc               | no        | Uses <code>io[ ]</code> . See (1)                                                                                                       |
| putchar            | no        | Uses <code>io[ ]</code> . See (1)                                                                                                       |
| puts               | no        | Uses <code>io[ ]</code> . See (1)                                                                                                       |
| qsort              | yes       | —                                                                                                                                       |
| raise              | no        | Updates the signal handler table                                                                                                        |
| rand               | no        | Uses static variable to remember latest random number. Must diverge from ANSI standard to define reentrant <code>rand</code> . See (4). |
| read               | no        | Calls <code>_read</code>                                                                                                                |
| <code>_read</code> | no        | Uses global File System Simulation buffer, <code>_fss_buffer</code>                                                                     |
| realloc            | no        | See <code>malloc</code> (5).                                                                                                            |
| remove             | —         | Skeleton only.                                                                                                                          |
| rename             | —         | Skeleton only.                                                                                                                          |
| rewind             | —         | Skeleton only.                                                                                                                          |
| sbrk               | no        | Allocates memory which is assigned at <code>locate</code> time. Needs kernel for memory management.                                     |
| scanf              | no        | Uses <code>io[ ]</code> , calls <code>_doscan</code> . See (1).                                                                         |
| setbuf             | no        | Sets <code>io[ ]</code> . See (1).                                                                                                      |
| setjmp             | yes       | —                                                                                                                                       |
| setlocale          | —         | Skeleton function                                                                                                                       |

| Function | Reentrant | Cause                                                                                                             |
|----------|-----------|-------------------------------------------------------------------------------------------------------------------|
| setvbuf  | no        | Sets iob and calls malloc. See (1) / (5).                                                                         |
| signal   | no        | Updates the signal handler table                                                                                  |
| sin      | yes       | —                                                                                                                 |
| sinh     | no        | Sinh calls exp() which sets errno. If errno is discarded sinh is reentrant.                                       |
| sprintf  | no        | Calls dprintf. See (1).                                                                                           |
| sqrt     | no        | Sets errno. See (2).                                                                                              |
| srand    | no        | See rand                                                                                                          |
| sscanf   | no        | Calls _doscan                                                                                                     |
| stat     | no        | Uses global File System Simulation buffer, _fss_buffer                                                            |
| strcat   | yes       | —                                                                                                                 |
| strchr   | yes       | —                                                                                                                 |
| strcmp   | yes       | —                                                                                                                 |
| strcoll  | —         | Skeleton function                                                                                                 |
| strcpy   | yes       | —                                                                                                                 |
| strcspn  | yes       | —                                                                                                                 |
| strerror | yes       | —                                                                                                                 |
| strftime | yes       | —                                                                                                                 |
| strlen   | yes       | —                                                                                                                 |
| strncat  | yes       | —                                                                                                                 |
| strncmp  | yes       | —                                                                                                                 |
| strncpy  | yes       | —                                                                                                                 |
| strpbrk  | yes       | —                                                                                                                 |
| strrchr  | yes       | —                                                                                                                 |
| strspn   | yes       | —                                                                                                                 |
| strstr   | yes       | —                                                                                                                 |
| strtod   | yes       | —                                                                                                                 |
| strtok   | no        | Strtok saves last position in string in local static variable. This function is not reentrant by design. See (4). |
| strtol   | no        | Sets errno. See (2).                                                                                              |
| strtoul  | no        | Sets errno. See (2).                                                                                              |

| Function | Reentrant | Cause                                                                                                                       |
|----------|-----------|-----------------------------------------------------------------------------------------------------------------------------|
| strxfrm  | –         | Skeleton function                                                                                                           |
| system   | –         | Skeleton function                                                                                                           |
| tan      | no        | Sets errno. See (2).                                                                                                        |
| tanh     | no        | Uses sinh for calculation.                                                                                                  |
| time     | no        | Uses static variable which defines initial start time                                                                       |
| tmpfile  | no        | Uses iob[ ]. See (1).                                                                                                       |
| tmpnam   | no        | Uses local buffer to build filename. Function can be adapted to use user buffer. This makes the function non ANSI. See (4). |
| toascii  | yes       | –                                                                                                                           |
| tolower  | yes       | –                                                                                                                           |
| toupper  | yes       | –                                                                                                                           |
| ungetc   | no        | Uses static buffer to hold ungetted characters for each file. Can be moved into iob structure. See (1).                     |
| unlink   | no        | Calls _unlink                                                                                                               |
| _unlink  | no        | Uses global File System Simulation buffer, _fss_buffer                                                                      |
| vfprintf | no        | Uses iob[ ], calls doprint. See (1).                                                                                        |
| vprintf  | no        | Uses iob[ ], calls doprint. See (1).                                                                                        |
| vsprintf | no        | Calls doprint.                                                                                                              |
| wcstombs | –         | Skeleton function                                                                                                           |
| wctomb   | –         | Skeleton function                                                                                                           |
| write    | no        | Calls _write                                                                                                                |
| _write   | no        | Uses global File System Simulation buffer, _fss_buffer                                                                      |

Table 6-4: C library reentrancy

Several functions in the C library are not reentrant due to the following reasons:

- The iob[ ] structure is static. This influences all I/O functions.
- The ungetc[ ] array is static. This array holds the characters (one for each stream) when ungetc( ) is called.

- The variable `errno` is globally defined. The following functions read or modify `errno`:  
`acos, asin, _doprint, _doscan, exp, fgetpos, fsetpos, ftell, log, log10, perror, pow, rewind, sqrt, strerror, strtol, strtoul, tan`
- `_doprint` and `_doscan` use static variables for e.g. character counting in strings.
- Some string functions use locally defined (static) buffers. This is prescribed by ANSI.
- `malloc` uses a static heap space.

The following description discusses these items into more detail. The numbers at the begin of each paragraph relate to the number references in the table above.

### **(1) *job structures***

The I/O part of the C library is not reentrant by design. This is mainly caused by the static declaration of the `job[]` array. The functions which use elements of this array access these elements via pointers ( `FILE *` ).

Building a multi-process system that is created in one link-run is hard to do. The C language scoping rules for external variables make it difficult to create a private copy of the `job[]` array. Currently, the `job[]` array has external scope. Thus it is visible in every module involved in one link phase. If these modules comprise several tasks (processes) in a system each of which should have its private copy of `job[]`, it is apparent that the `job[]` declaration should be changed. This requires adaption of the library to the multi-tasking environment. The library modules must use a process identification as an index for determining which `job[]` array to use. Thus the library is suitable for interfacing to that kernel only.

Another approach for the `job[]` declaration problem is to declare the array static in one of the modules which create a task. Thus there can be more than one `job[]` array in the system without having conflicts at link time. This brings several restrictions: Only the module that holds the declaration of the static `job[]` can use the standard file handles `stdin`, `stdout` and `stderr` (which are the first three entries in `job[]`). Thus all I/O for these three file handles should be located in one module.

## (2) *errno declaration*

Several functions in the C library set the global variable `errno`. After completion of the function the user program may consult this variable to see if some error occurred. Since most of the functions that set `errno` already have a return type (this is the reason for using `errno`) it is not possible to check successful completion via the return type.

The library routines can set `errno` to the values defined in `errno.h`. See the file `errno.h` for more information.

`errno` can be set to `ERR_FORMAT` by the print and scan functions in the C library if you specify illegal format strings.

`errno` can be set to `ERR_NOFLOAT` by the scan functions if you use floating point formatting while using the `SMALL` formatting routines. See also the next section *Printf and Scanf Formatting Routines*.

`errno` will never be set to `ERR_NOLONG` or `ERR_NOPOINT` since the Tricore C library supports long and pointer conversion routines for input and output.

`errno` can be set to `ERANGE` by the following functions: `exp()`, `strtol()`, `strtoul()` and `tan()`. These functions may produce results that are out of the valid range for the return type. If so, the result of the function will be the largest representable value for that type and `errno` is set to `ERANGE`.

`errno` is set to `EDOM` by the following functions: `acos()`, `asin()`, `log()`, `pow()` and `sqrt()`. If the arguments for these functions are out of their valid range (e.g. `sqrt(-1)`), `errno` is set to `EDOM`.

`errno` can be set to `ERR_POS` by the file positioning functions `ftell()`, `fsetpos()` and `fgetpos()`.

## (3) *ungetc*

Currently the `ungetc` buffer is static. For each file entry in the `iob[]` structure array, there is one character available in the buffer to `ungetc` a character.

#### (4) *local buffers*

`tmpnam()` creates a temporary filename and returns a pointer to a local static buffer. This is according to the ANSI definition. Changing this function such that it creates the name in a user specified buffer requires another calling interface. Thus the function would be no longer portable.

`strtok()` scans through a string and remembers that the string and the position in the string for subsequent calls. This function is not reentrant by design. Making it reentrant requires support of a kernel to store the information on a per process basis.

`rand()` generates a sequence of random numbers. The function uses the value returned by a previous call to generate the next value in the sequence. This function can be made reentrant by specifying the previous random value as one of the arguments. However, then it is no longer a standard function.

#### (5) *malloc*

Malloc uses a heap space which is assigned at locate time. Thus this implementation is not reentrant. Making a reentrant malloc requires that the `sbrk()` function can do some sort of system call to obtain free memory space on a per process basis. This is not easy to solve within the current context of the library. This requires adaption to a kernel.



This paragraph on reentrancy applies to multi-process environments only. If reentrancy is required for calling library functions from an exception handler, another approach is required. For such a situation it is of no use to allocate e.g. multiple `job[]` structures. In such a situation several pieces of code in the library have to be declared 'atomic': this means that interrupts have to be disabled while executing an atomic piece of code.



## **6.4 RUN-TIME LIBRARY**

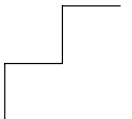
Some compiler generated code contains calls to run-time library functions that would use too much code if generated as inline code. The name of a run-time library function always contains two leading underscores.

The run-time library functions are included in the C library (`libc.a` or `libcs.a`).

# CHAPTER 7

## **RUN-TIME ENVIRONMENT**

---



---

# 7 | CHAPTER

---

## 7.1 STARTUP CODE

When linking your C modules with the library, you automatically link the object module, containing the C startup code. This module is called `cstart.obj` and is present in every C library.

Because this module specifies the run-time environment of your TriCore C application, you might want to edit it to match your needs. Therefore, this module is delivered in source in the file `cstart.asm` in the `src` subdirectory of the `lib` directory. Typically, you will copy the template startup file to your own directory and edit it. The startup code contains equates to tune the startup code. The invocation (using the **cctri** control program) is:

```
cctri -c cstart.asm
```

In the C startup code an absolute code section is defined for setting up the power on vector and the TriCore C environment. The power-on vector contains a definition of the `_START` label. This global label should not be removed, since the C compiler refers to it. It is also used as the default start address of the application.

In the file `cstart.asm` the actual location of several special function registers is required. These addresses are specified in the `regcpu_name.def` SFR system include files. You can include such a file with the assembler option **-Ccpu\_name**. In EDE the appropriate file is included when you have selected a CPU type. If you do not specify an SFR file, the default SFR `regtc10gp.def` file is included.

The stack size is defined in the locator control file (`tri.i` in directory `etc`) with the macros `USTACK` and `ISTACK` which results in sections called `ustack` and `istack`. See section 7.3, *Stack* for detailed information on the stack.

The heap is defined in the description file with the keyword `heap`, which results in a section called `heap`. See section 7.4 *Heap* for detailed information on heap management.

The startup code takes care of clearing global variables and initializing C variables residing in RAM. The startup code copies the initial values of initialized C variables from ROM to RAM, using a locator generated table (also known as the 'copy table') and a run-time library function `_c_init`.

When everything described above has been executed, your C application is called, using the global label `main`, which has been generated by **ctri** for the C function `main()`.

When the C application 'returns', which is not likely to happen in an embedded environment, the program ends with a `DEBUG16` instruction, at the assembly label `_exit`. When using a debugger, it can be useful to set a breakpoint on this label to indicate that the program has reached the end, or that the library function `exit()` has been called.

To control `cstart.asm` from within EDE, you first have to add `cstart.asm` to your project:



Select the **Project | Configure Selected CPU...** menu item and activate the **Cstart** tab. Enable the **Automatically Add cstart.asm** check box and click **OK**.

The file `cstart.asm` is added to your project. Now you can specify all your startup settings in the **Startup Code** tab of the **Configure Selected CPU** dialog:



Select the **Project | Configure Selected CPU...** menu item.

You can specify CPU settings in the same dialog:



Select the **Project | Configure Bus...** menu and select the appropriate bus configuration settings. EDE automatically defines macros according to the selected settings.

A number of other macro preprocessor symbols are used. These can be enabled or disabled using the assembler command line option **-D** with the following syntax:

**-D***Identifier*[=*replacement*]).

In the startup file the following macro preprocessor symbols are used:

| Define                                              | Description                         |
|-----------------------------------------------------|-------------------------------------|
| <b>External Boot Memory Configuration (BOOTCFG)</b> |                                     |
| <code>_BOOTCFG_ADDR</code>                          | Address generation value            |
| <code>_BOOTCFG_AGEN</code>                          | Read access wait-states value       |
| <code>_BOOTCFG_BCGEN</code>                         | Address Cycles value                |
| <code>_BOOTCFG_CFG</code>                           | Variable wait-state insertion value |

| Define                                 | Description                                                                                                                    |
|----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| _BOOTCFG_CMULT                         | Extended address setup value                                                                                                   |
| _BOOTCFG_SETUP                         | Active /WAIT level value                                                                                                       |
| _BOOTCFG_WAIT                          | Byte control signal timing mode value                                                                                          |
| _BOOTCFG_WAITINV                       | Wait cycle multiplier value                                                                                                    |
| _BOOTCFG_WAITRDC                       | Boot Memory Data Width value                                                                                                   |
| <b>Memory Control (PMUCON0/DMUCON)</b> |                                                                                                                                |
| _PMUCON0_CCBYP                         | Code cache bypass value                                                                                                        |
| _PMUCON0_CCSIZ                         | Code cache size value                                                                                                          |
| _DMUCON_DCAON                          | If defined, Enable data cache                                                                                                  |
| <b>Startup</b>                         |                                                                                                                                |
| _NO_BT_V_INIT                          | If define, Base Address of Trap Vector Table is not initialized with trap table start address (trap_tab).                      |
| _NO_BIV_INIT                           | If defined, Base Address of Interrupt Vector Table is not initialized with interrupt table start address (_lc_u_int_tab).      |
| _NO_ISP_INIT                           | If defined, Interrupt Stack Pointer is not initialized with end address of interrupt stack (_lc_ue_istack).                    |
| _NO_USP_INIT                           | If defined, User Stack Pointer is not initialized with end address of user stack (_lc_ue_istack).                              |
| _NO_PCX_RESET                          | If defined, the Previous Context is not explicitly cleared.                                                                    |
| _NO_PSW_RESET                          | If defined, the Call Depth Counter is not explicitly cleared.                                                                  |
| _NO_A0A1_ADDRESSING                    | If defined, global address register A0/A1 is not initialized with start address of the _a0/_a1 addressable area (_lc_gb_a0/1). |
| _NO_A8A9_ADDRESSING                    | If defined, global address register A8/A9 is not initialized with start address of the _a8/_a9 addressable area (_lc_gb_a8/9). |
| _NO_CSA_INIT                           | If defined, Context Save Area lists are not initialized.                                                                       |
| _NO_WATCHDOG_INIT                      | If defined, Watchdog timer disabled.                                                                                           |
| _NO_BUS_CONF                           | If defined, bus configuration registers are not initialized.                                                                   |
| _NO_C_INIT                             | If defined, C variables are not initialized.                                                                                   |

| Define                         | Description                                                                                                                                                                                                                                                                                    |
|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| _NO_ARG_INIT                   | If defined, disable initialization of argc and argv[].                                                                                                                                                                                                                                         |
| _NO_EXIT                       | If defined, C library function exit() or abort() not supported.                                                                                                                                                                                                                                |
| <b>Miscellaneous</b>           |                                                                                                                                                                                                                                                                                                |
| _CALL_INIT                     | Can be set to a function to be called before main. This function cannot have a return or arguments. This function can be used, for example, to initialize the serial port before main is called. This is useful for building programs without making any modifications to the original source. |
| _CALL_ENDINIT                  | Can be set to a function to be called before the ENDINIT instruction is executed. Like the CALLINIT function, it cannot not have a return value or arguments.                                                                                                                                  |
| <b>CPU functional bypasses</b> |                                                                                                                                                                                                                                                                                                |
| _TC112_XXX                     | If defined, TC112 CPU functional defect XXX is bypassed and/or checked.                                                                                                                                                                                                                        |
| _TC113_XXX                     | If defined, TC113 CPU functional defect XXX is bypassed and/or checked. See appendix C <i>CPU Functional Problems</i> CPU Functional Problems for a complete list of these macros.                                                                                                             |

Table 7-1: Defines used in *cstart.src*

The following table shows the locator labels used in the startup code.

| Define        | Description                                          |
|---------------|------------------------------------------------------|
| _START        | start label, mentioned in description file (tri.dsc) |
| _c_init       | label copy table init function                       |
| main          | start label user C program                           |
| exit          | start label of exit() function                       |
| _exit         | exit() function jumps to this place                  |
| _CALL_ENDINIT | label called before ENDINIT                          |
| _CALL_INIT    | _CALL_INIT label called before main()                |
| _lc_gb_a0     | locator label start of A0 addressable area           |
| _lc_gb_a1     | locator label start of A1 addressable area           |
| _lc_gb_a8     | locator label start of A8 addressable area           |

| Define        | Description                                |
|---------------|--------------------------------------------|
| _lc_gb_a9     | locator label start of A9 addressable area |
| _lc_u_int_tab | locator label interrupt table              |
| _lc_ub_csa    | locator label context save area begin      |
| _lc_ue_csa    | locator label context save area end        |
| _lc_ue_istack | locator label interrupt stack end          |
| _lc_ue_ustack | locator label user stack end               |

Table 7-2: Locator labels used in startup code

## 7.2 REGISTER USAGE

**ctri** will try to use the available registers as efficient as possible. The compiler uses a flexible register allocation scheme, which implies that any change to the C code may result in a different register usage.

The TriCore register file consists of 16 data registers and 16 address registers, which are 32 bits wide. The contents of registers D8–D15 and A10–A15 are saved by the CALL instruction and restored by the RET instruction. As a result, these registers (with the exception of the link register A11) can be used in a function without the need to save and restore their original contents. The registers D0–D7 and A2–A7 are considered "scratch": their contents is undefined after a function call. The "global registers" A0–A1 and A8–A9 are not changed by a function call or context switch.

For C function return types, the following registers are used:

| Return type       | Register | Description                   |
|-------------------|----------|-------------------------------|
| char              | D2       | return register               |
| short             | D2       |                               |
| int/ long / float | D2       |                               |
| double            | D2/D3    | (most significant part in D3) |
| pointer           | A2       | return register               |

Table 7-3: C function return types



Structures and unions of up to 8 bytes in size are returned in registers D2/D3. Larger structures or unions are returned on the stack. The address of this return area is passed as an implicit first argument in A4.

The following table summarize the register usage conventions used by **ctri**:

| Register | Usage                                  | Register | Usage                            |
|----------|----------------------------------------|----------|----------------------------------|
| D0       | scratch                                | A0       | global                           |
| D1       | scratch                                | A1       | global                           |
| D2       | return register for arithmetic types   | A2       | return register for pointers     |
| D3       | most significant part of 64 bit result | A3       | scratch                          |
| D4       | parameter                              | A4       | parameter                        |
| D5       | parameter                              | A5       | parameter                        |
| D6       | parameter                              | A6       | parameter                        |
| D7       | parameter                              | A7       | parameter                        |
| D8       | saved register                         | A8       | global                           |
| D9       | saved register                         | A9       | global                           |
| D10      | saved register                         | A10      | stack pointer                    |
| D11      | saved register                         | A11      | link register                    |
| D12      | saved register                         | A12      | saved register                   |
| D13      | saved register                         | A13      | saved register                   |
| D14      | saved register                         | A14      | saved register                   |
| D15      | saved register, implicit register      | A15      | saved register, implicit pointer |

Table 7-4: Register usage

### 7.3 STACK

The **stack** is used for local automatic variables, function parameters and saved registers.

The following diagram show the structure of a stack frame.

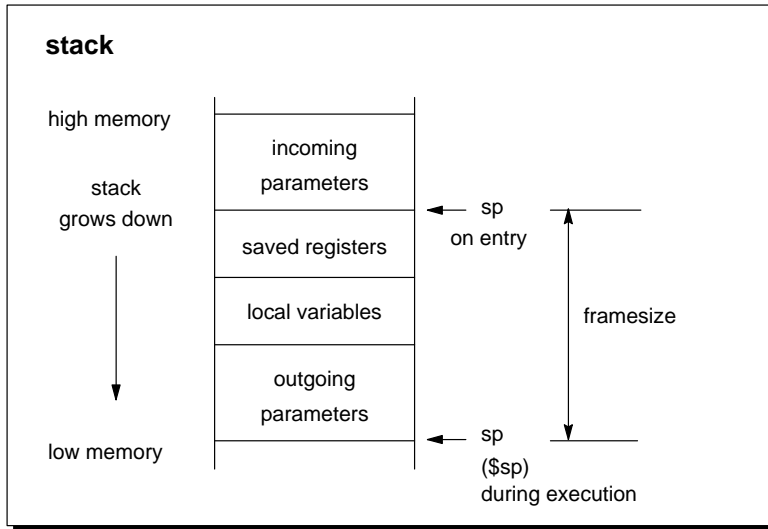


Figure 7-1: Stack diagram

The stack size is defined in the locator control file (`tri.i` in directory etc) with the macro `USTACK` and `istack`, which results in sections called `ustack` and `istack`.

The locator defined label `_lc_ue_ustack` refers to the top of the user stack area and is used in the file `cstart.asm` to initialize the user stack pointer register (SP). The locator defined label `_lc_ue_istack` refers to the top of the interrupt stack area and is used in the file `cstart.asm` to initialize the interrupt stack pointer register (ISP).

As long as the user program does not change the IS bit in the program status word (PSW), only the user stack is used. Refer to the *TriCore Architecture (v1.3) Manual* for the implications of an IS bit change.

## 7.4 HEAP

The heap is only needed when dynamic memory management library functions are used: `malloc()`, `calloc()`, `free()` and `realloc()`. The heap is a reserved area in memory. Only if you use one of the memory allocation functions listed above, the locator automatically allocates a heap, as specified in the locator description file with the keyword `heap`.

A special section called `heap` is used for the allocation of the heap area. The size of the heap is defined in the locator control file (`tri.i` in directory `etc`) with the macro `HEAP`, which results in a section called `heap`. The locator defined labels `_lc_bh` and `_lc_eh` (begin and end of heap) are used by the library function `sbrk()`, which is called by `malloc()` when memory is needed from the heap.



The special `heap` section is only allocated when its locator labels are used in the program.

## 7.5 FLOATING POINT ARITHMETIC

Floating point arithmetic support for the compiler **ctri** is included in the software as a separate set of libraries or in the hardware when available (only single precision). During linking you have to specify the desired floating point library after the C library. The libraries are reentrant, and only use temporary program stack memory.

To ensure portability of floating point arithmetic, floating point arithmetic for the compiler **ctri** has been implemented complying to the IEEE-754 standard for floating point arithmetic. See the *IEEE Standard Binary for Floating-Point Arithmetic* document [IEEE Computer Society, 1985] for more details on the floating point arithmetic definitions. This document is referred to as IEEE-754 in this manual.

The compiler **ctri** supports both single and double precision floating point operations using the ANSI C types `float` and `double` respectively. To optimize for speed, also a non-trapping library is included. For the library names, see section 6.3, *C Libraries*.

It is possible to intercept floating point exceptional cases and, if desired, handle them with an application defined exception handler. The intercepting of floating point exceptions is referred to as 'trapping'. Examples of how to install a trap handler are included.

### 7.5.1 COMPLIANCE TO IEEE-754

The level to which the floating point implementation complies to the IEEE-754 standard, depends on the chosen configuration.

All floating point calculations are executed using the 'round to nearest (even)' rounding mode, since this is required by ANSI-C 89. This is conform IEEE-754. Because there are no double precision floating point hardware instructions, an emulating library is always needed for double precision calculation.

When the use of hardware FPU instructions is chosen (**-FPU**), the available hardware instructions for single precision floating point will be used either in the compiler or in one of the libraries. For double precision floating point calculations the chosen floating point emulator library will be used. When no hardware FPU instructions are allowed, all floating point operations will be used from the chosen floating point emulator library.



In EDE you can specify to use the single precision floating point hardware: Select the **Project | C Compiler Options | Project Options...** menu item and enable the **Use hardware single precision floating point instructions** check box in the Misc tab.



This option is only available (and relevant) when you enable the **FPU present (on user defined CPU)** check box on the CPU tab in the **Project | Processor options...** menu item.



**-FPU** in Chapter 4, *Compiler Use*.

#### ***Compliance with IEEE-754: TriCore hardware FPU instructions***

The following implementation issues for the single precision hardware instructions (optionally implemented on the TriCore chip), are important:

- subnormals are not supported (hardware design decision).
- when converting single precision floats to integers, rounding is done to the nearest (even) integer. This does not comply with ANSI-C 89 or ISO-C 99, but does comply with IEEE-754, since this is the current rounding mode (hardware design decision).
- when a converted single precision float overflows the target integer type, the value is saturated to MAX\_INT or MIN\_INT (hardware design decision).

- whenever a double precision float is involved, the results are determined by the chosen emulation library.

#### ***Compliance with IEEE-754: Trapping emulation library***

The following implementation issues for the trapping floating point library are important:

- subnormals are not supported. This is conform the TriCore hardware design.
- when converting floats to integers, the value is truncated. This complies with ANSI-C 89 and ISO-C 99, but does not comply with IEEE-754, since the current rounding mode is 'round to nearest (even)'.
- when a converted float overflows the target integer type, a predictable value is assigned to the target integer.

#### ***Compliance with IEEE-754: Hand-optimized non-trapping emulation library***

The following implementation issues for the non-trapping floating point library are important:

- when calculating with floats, rounding is done to the nearest integer (rounding towards infinity when equally near).
- there is no distinction between -0 and +0
- when an operand of a calculation is a NaN, Inf or subnormal, the result is undefined.
- when the result of a calculation would be a subnormal, the result is 0.
- whenever a NaN or Inf would be the result of a calculation, the result is undefined
- when converting single precision floats to integers, rounding is done to the nearest integer (rounding towards infinity when equally near). This is similar to the TriCore FPU hardware.
- when converting double precision floats to integers, the value is truncated. This is similar to the trapping emulation library.
- when a converted float overflows the target integer type, the value is saturated to MAX\_INT or MIN\_INT.

### 7.5.2 SPECIAL FLOATING POINT VALUES

Below is a list of special, IEEE-754 defined, floating point values as they can occur during run-time.

| Special value            | Sign | Exponent  | Mantissa      |
|--------------------------|------|-----------|---------------|
| +0.0 (Positive Zero)     | 0    | all zeros | all zeros     |
| -0.0 (Negative Zero)     | 1    | all zeros | all zeros     |
| +INF (Positive Infinite) | 0    | all ones  | all zeros     |
| -INF (Negative Infinite) | 1    | all ones  | all zeros     |
| NaN (Not a number)       | 0    | all ones  | not all zeros |

Table 7-5: Special floating point values

### 7.5.3 TRAPPING FLOATING POINT EXCEPTIONS

Four floating point run-time libraries are delivered for every memory model:

with fp trap handling; without FPU instructions: `libfpt.a`

without trapping; without FPU instructions (**default**): `libfpn.a`

with fp trap handling; with FPU instructions: `libfpt_fpu.a`

without trapping; with FPU instructions: `libfpn_fpu.a`

By specifying the **-fptrap** option to the control program **cctri**, the trapping type floating point library is linked into your application. By specifying the **-FPU** option to the control program **cctri**, a floating point library with single precision FPU instructions is linked into your application. If these options are not specified, the floating point library without trapping mechanism and without FPU instructions is used.



In EDE you can specify to use the trapping type floating point library as follows: Select the Project | Linker/Locator Options... menu and enable the Use trapping floating point library check box in the Linker tab.

### ***IEEE-754 Trap Handler***

In the IEEE-754 standard a trap handler is defined, which is invoked on (specified) exceptional events, passing along much information about the event. To install your own trap handler, use the library call `_fp_install_trap_handler`. When installing your own exception handler, you must select on which types of exceptions you want to have your handler invoked, using the function call `_fp_set_exception_mask`. See below for more details on the floating point library exception handling function interface.

### ***SIGFPE Signal Handler***

In ANSI-C the regular approach of dealing with floating point exceptions is by installing a so-called signal handler by means of the ANSI-C library call `signal`. If such a handler is installed, floating point exceptions cause this handler to be invoked. To have the signal handler for the **SIGFPE** signal actually become operational with the provided floating point libraries, a (very) basic version of the IEEE-754 exception handler must be installed (see example below) which will raise the desired signal by means of the ANSI-C library function call `raise`. For this to be achieved, the function call `_fp_install_trap_handler` is present. When installing your own exception handler, you will have to select on which types of exceptions you want to receive a signal, using the function call `_fp_set_exception_mask`. See further below for more details on the floating point library exception handling function interface.

There is no way to specify any information about the context or nature of the exception to the signal handler. Just that a floating point exception occurred can be detected. See therefor the IEEE-754 trap handler discussion above if you want more control over floating point results.

Example:

```
#include <float.h>
#include <signal.h>

static void pass_fp_exception_to_signal(
 _fp_exception_info_t *info)
{
 info; /* suppress parameter not used warning */

 /* cause SIGFPE signal to be raised */

 raise(SIGFPE);
 /*
 * now continue the program
 * with the unaltered result
 */
}
```

#### **7.5.4 FLOATING POINT TRAP HANDLING API**

For purposes of dealing with floating point arithmetic exceptions, the following library calls are available:

```
#include <float.h>

int _fp_get_exception_mask(void);
void _fp_set_exception_mask(int);
```

A pair of functions to get or set the mask which controls which type of floating point arithmetic exceptions are either ignored or passed on to the trap handler. The types of possible exception flag bits are defined as:

```
EFINVOP
EFDIVZ
EFOVFL
EFUNFL
EFINEXCT
```

while,

```
EFALL
```

is the OR of all possible flags. See below for an explanation of each flag.



```
#include <float.h>

int _fp_get_exception_status(void);
void _fp_set_exception_status(int);
```

A pair of functions for examining or presetting the status word containing the accumulation of all floating point exception types which occurred so far. See the possible exception type flags above.

```
#include <float.h>

void _fp_install_trap_handler(void (*)
 (_fp_exception_info_t *));
```

This function call expects a pointer to a function, which in turn expects a pointer to a structure of type `_fp_exception_info_t`. The members of `_fp_exception_info_t` are:

#### **exception**

This member contains one of the following (numerical) values:

```
EFINVOP
EFDIVZ
EFOVFL
EFUNFL
EFINEXCT
```

#### **operation**

This member contains one of the following numbers:

```
_OP_ADDITION
_OP_SUBTRACTION
_OP_COMPARISON
_OP_EQUALITY
_OP_LESS_THAN
_OP_LARGER_THAN
_OP_MULTIPLICATION
_OP_DIVISION
_OP_CONVERSION
```

```
source_format
destination_format
```

Numerical values of these two members are:

```

_TYPE_SIGNED_CHARACTER
_TYPE_UNSIGNED_CHARACTER
_TYPE_SIGNED_SHORT_INTEGER
_TYPE_UNSIGNED_SHORT_INTEGER
_TYPE_SIGNED_INTEGER
_TYPE_UNSIGNED_INTEGER
_TYPE_SIGNED_LONG_INTEGER
_TYPE_UNSIGNED_LONG_INTEGER
_TYPE_FLOAT
_TYPE_DOUBLE

```

```

operand1 /* left side of binary or */
 /* right side of unary */
operand2 /* right side for binary */
result

```

These three are of the following type, to receive and return a value of arbitrary type:

```

typedef union _fp_value_union_t
{
 char c;
 unsigned char uc;
 short s;
 unsigned short us;
 int i;
 unsigned int ui;
 long l;
 unsigned long ul;
 float f;
 #if ! _SINGLE_FP
 double d;
 #endif
}
_fp_value_union_t;

```



The member `d` is not present when specifying the **-F** option to the C compiler.



The following table lists all the exception code flags, the corresponding error description and result:

| Error Description | Exception Flag                                                                                                            | Default Result with Trapping |
|-------------------|---------------------------------------------------------------------------------------------------------------------------|------------------------------|
| Invalid Operation | EFINVOP                                                                                                                   | NaN                          |
| Division by zero  | EFDIVZ                                                                                                                    | +INF or -INF                 |
| Overflow          | EFOVFL                                                                                                                    | +INF or -INF                 |
| Underflow         | EFUNFL                                                                                                                    | zero                         |
| Inexact           | EFINEXT                                                                                                                   | undefined                    |
| INF               | Infinite which is the largest absolute floating point number, which is always: $-INF < \text{every finite number} < +INF$ |                              |
| NAN               | Not a Number, a symbolic entity encoded in floating point format.                                                         |                              |

Table 7-6: Exception Type Flag Codes

To ensure all exception types are specified, you can specify **EFALL** to a function, which is the binary OR of all above enlisted flags.

# APPENDIX

## A

### **FLEXIBLE LICENSE MANAGER (FLEXlm)**

---



---

**A**

**APPENDIX**

---

## **1 INTRODUCTION**

This appendix discusses Globetrotter Software's Flexible License Manager and how it is integrated into the TASKING toolchain. It also contains descriptions of the Flexible License Manager license administration tools that are included with the package, the daemon log file and its contents, and the use of daemon options files to customize your use of the TASKING toolchain.

## **2 LICENSE ADMINISTRATION**

### **2.1 OVERVIEW**

The Flexible License Manager (FLEXlm) is a set of utilities that, when incorporated into software such as the TASKING toolchain, provides for managing access to the software.

The following terms are used to describe FLEXlm concepts and software components:

|               |                                                                                                                                                                                                                                   |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| feature       | A feature could be any of the following: <ul style="list-style-type: none"><li>• A TASKING software product.</li><li>• A software product from another vendor.</li></ul>                                                          |
| license       | The right to use a feature. FLEXlm restricts licenses for features by counting the number of licenses for features in use when new requests are made by the application software.                                                 |
| client        | A TASKING application program.                                                                                                                                                                                                    |
| daemon        | A process that "serves" clients. Sometimes referred to as a <i>server</i> .                                                                                                                                                       |
| vendor daemon | The daemon that dispenses licenses for the requested features. This daemon is built by an application's vendor, and contains the vendor's personal encryption code. <b>Tasking</b> is the vendor daemon for the TASKING software. |

license daemon

The daemon process that sends client processes to the correct vendor daemon on the correct machine. The same license daemon is used by all applications from all vendors, as this daemon neither performs encryption nor dispenses licenses. The license daemon processes no user requests on its own, but forwards these requests to other daemons (the vendor daemons).

**server node** A computer system that is running both the license and vendor daemon software. The server node will contain all the dynamic information regarding the usage of all the features.

**license file** An end-user specific file that contains descriptions of the server nodes that can run the license daemons, the various vendor daemons, and the restrictions for all the licensed features.

The TASKING software is granted permission to run by FLEXlm daemons; the daemons are started when the TASKING toolchain is installed and run continuously thereafter. Information needed by the FLEXlm daemons to perform access management is contained in a license data file that is created during the toolchain installation process. As part of their normal operation, the daemons log their actions in a daemon log file, which can be used to monitor usage of the TASKING toolchain.

The following sections discuss:

- Installation of the FLEXlm daemons to provide for access to the TASKING toolchain.
- Customizing your use of the toolchain through the use of a daemon options file.
- Utilities that are provided to assist you in performing license administration functions.
- The daemon log file and its contents.

For additional information regarding the use of FLEXlm, refer to the chapter *Software Installation*.

## 2.2 PROVIDING FOR UNINTERRUPTED FLEXLM OPERATION

TASKING products licensed through FLEXlm contain a number of utilities for managing licenses. These utilities are bundled in the form of an extra product under the name SW000098. TASKING products themselves contain two additional files for FLEXlm in a *flexlm* subdirectory:

|             |                                     |
|-------------|-------------------------------------|
| Tasking     | The Tasking daemon (vendor daemon). |
| license.dat | A template license file.            |

If you have already installed FLEXlm (e.g. as part of another product) then it is not needed to install the bundled SW000098. After installing SW000098 on UNIX, the directory `/usr/local/flexlm` will contain two subdirectories, `bin` and `licenses`. After installing SW000098 on Windows the directory `c:\flexlm` will contain the subdirectory `bin`. The exact location may differ if FLEXlm has already been installed as part of a non-TASKING product but in general there will be a directory for executables such as `bin`. That directory must contain a copy of the **Tasking** daemon shipped with every TASKING product. It also contains the files:

|       |                                                     |
|-------|-----------------------------------------------------|
| lmgrd | The FLEXlm daemon (license daemon).                 |
| lm*   | A group of FLEXlm license administration utilities. |

Next to it, a license file must be present containing the information of all licenses. This file is usually called `license.dat`. The default location of the license file is in directory `c:\flexlm` for Windows and in `/usr/local/flexlm/licenses` for UNIX. If you did install SW000098 then the `licenses` directory on UNIX will be empty, and on Windows the file `license.dat` will be empty. In that case you can copy the `license.dat` file from the product to the `licenses` directory after filling in the data from your "License Information Form".



Be very careful not to overwrite an existing `license.dat` file because it contains valuable data.

Example `license.dat`:

```
SERVER HOSTNAME HOSTID PORT
DAEMON Tasking /usr/local/flexlm/bin/Tasking
FEATURE SW008002-32 Tasking 3.000 EXPDATE NUSERS PASSWORD SERIAL
```



After modifications from a license data sheet (example):

```
SERVER elliot 5100520c 7594
DAEMON Tasking /usr/local/flexlm/bin/Tasking
FEATURE SW008002-32 Tasking 3.000 1-jan-00 4 0B1810310210A6894 "123456"
```

If the `license.dat` file already exists then you should make sure that it contains the DAEMON and FEATURE lines from your license data sheet. An appropriate SERVER line should already be present in that case. You should only add a new SERVER line if no SERVER line is present. The third field of the DAEMON line is the pathname to the **Tasking** daemon and you may change it if necessary.

The default location for the license file on Windows is:

```
c:\flexlm\license.dat
```

On UNIX this is:

```
/usr/local/flexlm/licenses/license.dat
```

If the pathname of the resulting license file differs from this default location then you must set the environment variable **LM\_LICENSE\_FILE** to the correct pathname. If you have more than one product using the FLEXlm license manager you can specify multiple license files by separating each pathname (*lfp<sub>path</sub>*) with a ';' (on UNIX also ':') :

Windows:

```
set LM_LICENSE_FILE=lfppath;lfppath...
```

UNIX:

```
setenv LM_LICENSE_FILE lfppath:lfppath...
```

If you are running the TASKING software on multiple nodes, you have three options for making your license file available on all the machines:

1. Place the license file in a partition which is available (via NFS on Unix systems) to all nodes in the network that need the license file.
2. Copy the license file to all of the nodes where it is needed.
3. Set LM\_LICENSE\_FILE to "*port@host*", where *host* and *port* come from the SERVER line in the license file.

When the main license daemon **lmgrd** already runs it is sufficient to type the command:

```
lmreread
```

for notifying the daemon that the `license.dat` file has been changed. Otherwise, you must type the command:

```
lmgrd >/usr/tmp/license.log &
```

Both commands reside in the flexlm bin directory mentioned before.

## 2.3 DAEMON OPTIONS FILE

It is possible to customize the use of TASKING software using a daemon options file. This options file allows you to reserve licenses for specified users or groups of users, to restrict access to the TASKING toolchain, and to set software timeouts. The following table lists the keywords that are recognized at the start of a line of a daemon options file.

| Keywords | Function                                                                                                            |
|----------|---------------------------------------------------------------------------------------------------------------------|
| RESERVE  | Ensure that TASKING software will always be available to one or more users or on one or more host computer systems. |
| INCLUDE  | Specify a list of users who are allowed exclusive access to the TASKING software.                                   |
| EXCLUDE  | Specify a list of users who are not allowed to use the TASKING software.                                            |
| GROUP    | Specify a group of users for use in the other commands.                                                             |
| TIMEOUT  | Allow licenses that are idle for a specified time to be returned to the free pool, for use by someone else.         |
| NOLOG    | Causes messages of the specified type to be filtered out of the daemon's log output.                                |

*Table A-1: Daemon options file keywords*

In order to use the daemon options capability, you must create a daemon options file and list its pathname as the fourth field on the **DAEMON** line for the **Tasking** daemon in the license file. For example, if the daemon options were in file `/usr/local/flexlm/Tasking.opt` (UNIX), then you would modify the license file **DAEMON** line as follows:

```
DAEMON Tasking /usr/local/Tasking /usr/local/flexlm/Tasking.opt
```

A daemon options file consists of lines in the following format:

```
RESERVE number feature {USER | HOST | DISPLAY | GROUP} name
INCLUDE feature {USER | HOST | DISPLAY | GROUP} name
EXCLUDE feature {USER | HOST | DISPLAY | GROUP} name
GROUP name <list_of_users>
TIMEOUT feature timeout_in_seconds
NOLOG {IN | OUT | DENIED | QUEUED}
REPORTLOG file
```

Lines beginning with the sharp character (#) are ignored, and can be used as comments. For example, the following options file would reserve one copy of feature SWxxxxxx-xx for user “pat”, three copies for user “lee”, and one copy for anyone on a computer with the hostname of “terry”; and would cause QUEUED messages to be omitted from the log file. In addition, user “joe” and group “pinheads” would not be allowed to use the feature SWxxxxxx-xx:

```
GROUP pinheads moe larry curley
RESERVE 1 SWxxxxxx-xx USER pat
RESERVE 3 SWxxxxxx-xx USER lee
RESERVE 1 SWxxxxxx-xx HOST terry
EXCLUDE SWxxxxxx-xx USER joe
EXCLUDE SWxxxxxx-xx GROUP pinheads
NOLOG QUEUED
```

### 3 LICENSE ADMINISTRATION TOOLS

The following utilities are provided to facilitate license management by your system administrator. In certain cases, execution access to a utility is restricted to users with root privileges. Complete descriptions of these utilities are provided at the end of this section.

#### ***lmcksum***

Prints license checksums.

#### ***lmdiag*** (Windows only)

Diagnoses license checkout problems.

#### ***lmdown***

Gracefully shuts down all license daemons (both **lmgrd** all vendor daemons, such as **Tasking**) on the license server.

***lmgrd***

The main daemon program for FLEXlm.

***lmbostid***

Reports the hostid of a system.

***lmremove***

Removes a single user's license for a specified feature.

***lmreread***

Causes the license daemon to reread the license file and start any new vendor daemons.

***lmstat***

Helps you monitor the status of all network licensing activities.

***lmswitchr***

Switches the report log file.

***lmver***

Reports the FLEXlm version of a library or binary file.

***lmtools*** (*Windows only*)

This is a graphical Windows version of the license administration tools.

### 3.1 LMCKSUM

#### Name

**lmcksum** – print license checksums

#### Synopsis

**lmcksum** [ **-c** *license\_file* ] [ **-k** ]

#### Description

The **lmcksum** program will perform a checksum of a license file. This is useful to verify data entry errors at your location. **lmcksum** will print a line-by-line checksum for the file as well as an overall file checksum.

The following fields participate in the checksum:

- hostid on the SERVER lines
- daemon name on the DAEMON lines
- feature name, version, daemon name, expiration date, # of licenses, encryption code, vendor string and hostid on the FEATURE lines
- daemon name and encryption code on FEATURESET lines

#### Options

**-c** *license\_file*

Use the specified *license\_file*. If no **-c** option is specified, **lmcksum** looks for the environment variable LM\_LICENSE\_FILE in order to find the license file to use. If that environment variable is not set, **lmcksum** looks for the file c:\flexlm\license.dat (Windows), or /usr/local/flexlm/licenses/license.dat (UNIX).

**-k**

Case-sensitive checksum. If this option is specified, **lmcksum** will compute the checksum using the exact case of the FEATURE's and FEATURESET's encryption code.

## 3.2 LMDIAG (Windows only)

### Name

**lmdiag** – diagnose license checkout problems

### Synopsis

**lmdiag** [ **-c** *license\_file* ] [ **-n** ] [*feature* ]

### Description

**lmdiag** (Windows only) allows you to diagnose problems when you cannot check out a license.

If no *feature* is specified, **lmdiag** will operate on all features in the license file(s) in your path. **lmdiag** will first print information about the license, then attempt to check out each license. If the checkout succeeds, **lmdiag** will indicate this. If the checkout fails, **lmdiag** will give you the reason for the failure. If the checkout fails because **lmdiag** cannot connect to the license server, then you have the option of running "extended connection diagnostics".

These extended diagnostics attempt to connect to each port on the license server node, and can detect if the port number in the license file is incorrect. **lmdiag** will indicate each port number that is listening, and if it is an **lmgrd** process, **lmdiag** will indicate this as well. If **lmdiag** finds the vendor daemon for the feature being tested, then it will indicate the correct port number for the license file to correct the problem.

### Parameters

*feature*      Diagnose this feature only.

### Options

**-c** *license\_file*

Diagnose the specified *license\_file*. If no **-c** option is specified, **lmdiag** looks for the environment variable `LM_LICENSE_FILE` in order to find the license file to use. If that environment variable is not set, **lmdiag** looks for the file `c:\flexlm\license.dat` (Windows), or `/usr/local/flexlm/licenses/license.dat` (UNIX).

**-n**

Run in non-interactive mode; **lmdiag** will not prompt for any input in this mode. In this mode, extended connection diagnostics are not available.

### 3.3 LMDOWN

#### Name

**lmdown** – graceful shutdown of all license daemons

#### Synopsis

**lmdown** [ **-c** *license\_file* ] [ **-q** ]

#### Description

The **lmdown** utility allows for the graceful shutdown of all license daemons (both **lmgrd** and all vendor daemons, such as **Tasking**) on all nodes. You may want to protect the execution of **lmdown**, since shutting down the servers causes users to lose their licenses. See the **-p** option in Section 3.4, **lmgrd**.

**lmdown** sends a message to every license daemon asking it to shut down. The license daemons write out their last messages to the log file, close the file, and exit. All licenses which have been given out by those daemons will be revoked, so that the next time a client program goes to verify his license, it will not be valid.

#### Options

**-c** *license\_file*

Use the specified *license\_file*. If no **-c** option is specified, **lmdown** looks for the environment variable **LM\_LICENSE\_FILE** in order to find the license file to use. If that environment variable is not set, **lmdown** looks for the file `c:\flexlm\license.dat` (Windows), or `/usr/local/flexlm/licenses/license.dat` (UNIX).

**-q**

Quiet mode. If this switch is not specified, **lmdown** asks for confirmation before asking the license daemons to shut down. If this switch is specified, **lmdown** will not ask for confirmation.



lmgrd, lmstat, lmread

### 3.4 LMGRD

#### Name

**lmgrd** – flexible license manager daemon

#### Synopsis

**lmgrd** [ **-c** *license\_file* ] [ **-l** *logfile* ] [ **-2 -p** ] [ **-t** *timeout* ] [ **-s** *interval* ]

#### Description

**lmgrd** is the main daemon program for the FLEXlm distributed license management system. When invoked, it looks for a license file containing all required information about vendors and features. On UNIX systems, it is strongly recommended that **lmgrd** be run as a non-privileged user (not root).

#### Options

**-c** *license\_file*

Use the specified *license\_file*. If no **-c** option is specified, **lmgrd** looks for the environment variable LM\_LICENSE\_FILE in order to find the license file to use. If that environment variable is not set, **lmgrd** looks for the file `c:\flexlm\license.dat` (Windows), or `/usr/local/flexlm/licenses/license.dat` (UNIX).

**-l** *logfile*

Specifies the output log file to use. Instead of using the **-l** option you can use output redirection (**>** or **>>**) to specify the name of the output log file.

**-2 -p**

Restricts usage of **lmdown**, **lmreread**, and **lmremove** to a FLEXlm administrator who is by default root. If there is a UNIX group called "lmadmin" then use is restricted to only members of that group. If root is not a member of this group, then root does not have permission to use any of the above utilities.

**-t** *timeout*

Specifies the *timeout* interval, in seconds, during which the license daemon must complete its connection to other daemons if operating in multi-server mode. The default value is 10 seconds. A larger value may be desirable if the daemons are being run on busy systems or a very heavily loaded network.



**-s *interval*** Specifies the log file timestamp *interval*, in minutes. The default is 360 minutes. This means that every six hours **lmgrd** logs the time in the log file.



lmdown, lmstat

### **3.5 LMHOSTID**

#### **Name**

**lmhostid** – report the hostid of a system

#### **Synopsis**

**lmhostid**

#### **Description**

**lmhostid** calls the FLEXlm version of `gethostid` and displays the results.

The output of **lmhostid** looks like this:

```
lmhostid - Copyright (C) 1989, 1999 Globetrotter Software, Inc.
The FLEXlm host ID of this machine is "1200abcd"
```

#### **Options**

**lmhostid** has no command line options.

## 3.6 LMREMOVE

### Name

**lmremove** – remove specific licenses and return them to license pool

### Synopsis

**lmremove** [ **-c** *license\_file* ] *feature user host* [ *display* ]

### Description

The **lmremove** utility allows the system administrator to remove a single user's license for a specified feature. This could be required in the case where the licensed user was running the software on a node that subsequently crashed. This situation will sometimes cause the license to remain unusable. **lmremove** will allow the license to return to the pool of available licenses.

**lmremove** will remove all instances of “user” on node “host” on display “display” from usage of “feature”. If the optional **-c file** is specified, the indicated file will be used as the license file. Since removing a user's license can be disruptive, execution of **lmremove** is restricted to users with root privileges.

### Options

**-c** *license\_file*

Use the specified *license\_file*. If no **-c** option is specified, **lmremove** looks for the environment variable `LM_LICENSE_FILE` in order to find the license file to use. If that environment variable is not set, **lmremove** looks for the file `c:\flexlm\license.dat` (Windows), or `/usr/local/flexlm/licenses/license.dat` (UNIX).



lmstat

### 3.7 LMREREAD

#### Name

**lmreread** – tells the license daemon to reread the license file

#### Synopsis

**lmreread** [ **-c** *license\_file* ]

#### Description

**lmreread** allows the system administrator to tell the license daemon to reread the license file. This can be useful if the data in the license file has changed; the new data can be loaded into the license daemon without shutting down and restarting it.

The license administrator may want to protect the execution of **lmreread**. See the **-p** option in Section 3.4, *lmgrd* for details about securing access to **lmreread**.

**lmreread** uses the license file from the command line (or the default file, if none specified) only to find the license daemon to send it the command to reread the license file. The license daemon will always reread the file that it loaded from the original path. If you need to change the path to the license file read by the license daemon, then you must shut down the daemon and restart it with that new license file path.

You cannot use **lmreread** if the *SERVER* node names or port numbers have been changed in the license file. In this case, you must shut down the daemon and restart it in order for those changes to take effect.

**lmreread** does not change any option information specified in an options file. If the new license file specifies a different options file, that information is ignored. If you need to reread the options file, you must shut down (**lmdown**) the daemon and restart it.

#### Options

**-c** *license\_file*

Use the specified *license\_file*. If no **-c** option is specified, **lmreread** looks for the environment variable *LM\_LICENSE\_FILE* in order to find the license file to use. If that environment variable is not set, **lmreread** looks for the file *license.dat* in the default location.



**lmdown**

## 3.8 LMSTAT

### Name

**lmstat** – report status on license manager daemons and feature usage

### Synopsis

```
lmstat [-a] [-A] [-c license_file] [-f feature]
 [-l regular_expression] [-s server] [-S daemon] [-t timeout]
```

### Description

License administration is simplified by the **lmstat** utility. **lmstat** allows you to instantly monitor the status of all network licensing activities.

**lmstat** allows a system administrator to monitor license management operations including:

- Which daemons are running
- Users of individual features
- Users of features served by a specific DAEMON

### Options

**-a** Display all information.

**-A** List all active licenses.

**-c** *license\_file*

Use the specified *license\_file*. If no **-c** option is specified, **lmstat** looks for the environment variable `LM_LICENSE_FILE` in order to find the license file to use. If that environment variable is not set, **lmstat** looks for the file `c:\flexlm\license.dat` (Windows), or `/usr/local/flexlm/licenses/license.dat` (UNIX).

**-f** *feature* List all users of the specified *feature*(s).

**-l** *regular\_expression*

List all users of the features matching the given *regular\_expression*.

**-s** *server* Display the status of the specified *server* node(s).

**-S** *daemon* List all users of the specified *daemon*'s features.

- t *timeout*** Specifies the amount of time, in seconds, **lmstat** waits to establish contact with the servers. The default value is 10 seconds. A larger value may be desirable if the daemons are being run on busy systems or a very heavily loaded network.



lmgrd

### 3.9 LMSWITCHR (Windows only)

#### Name

**lmswitchr** – switch the report log file

#### Synopsis

**lmswitchr** [ **-c** *license\_file* ] *feature new-file*

or:

**lmswitchr** [ **-c** *license\_file* ] *vendor new-file*

#### Description

**lmswitchr** (Windows only) switches the report writer (REPORTLOG) log file. It will also start a new REPORTLOG file if one does not already exist.

#### Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>feature</i>  | Any feature this daemon supports.                        |
| <i>vendor</i>   | The name of the vendor daemon (such as <b>Tasking</b> ). |
| <i>new-file</i> | New file path.                                           |

#### Options

**-c** *license\_file* Use the specified *license\_file*. If no **-c** option is specified, **lmswitchr** looks for the environment variable LM\_LICENSE\_FILE in order to find the license file to use. If that environment variable is not set, **lmswitchr** looks for the file `c:\flexlm\license.dat` (Windows), or `/usr/local/flexlm/licenses/license.dat` (UNIX).

### **3.10 LMVER**

#### **Name**

**lmver** – report the FLEXlm version of a library or binary file

#### **Synopsis**

**lmver** *filename*

#### **Description**

The **lmver** utility reports the FLEXlm version of a library or binary file.

Alternatively, on UNIX systems, you can use the following commands to get the FLEXlm version of a binary:

**strings *file* | grep Copy**

#### **Parameters**

*filename*      Name of the executable of the product.



## **3.11 LICENSE ADMINISTRATION TOOLS FOR WINDOWS**

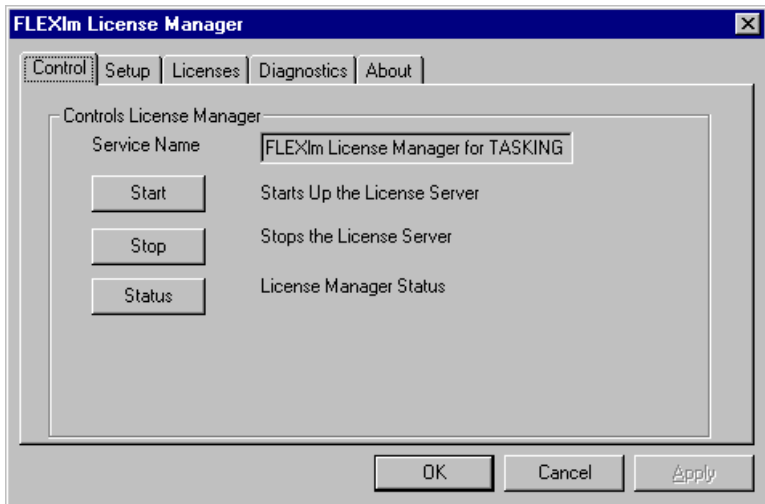
### **3.11.1 LMTOOLS FOR WINDOWS**

For the 32 Bit Windows Platforms, an **lmtools.exe** Windows program is provided. It has the same functionality as listed in the previous sections but is graphically-oriented. Simply run the program (Start | Programs | TASKING FLEXlm | FLEXlm Tools) and choose a button for the functionality required. Refer to the previous sections for information about the options of each feature. The command line interface is replaced by pop-up dialogs that can be filled out. The central EDIT field is where the license file path is placed. This will be used for all other functions and replaces the "**-c** *license\_file*" argument in the other utilities.

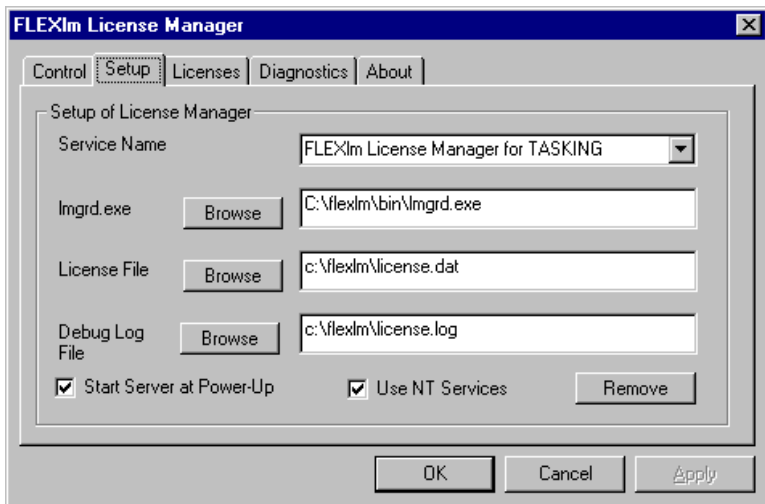
The HOSTID button displays the hostid's for the computer on which the program is running. The TIME button prints out the system's internal time settings, intended to diagnose any time zone problems. The TCP Settings button is intended to fix a bug in the Microsoft TCP protocol stack which has a symptom of very slow connections to computers. After pressing this button, the system will need to be rebooted for the settings to become effective.

### 3.11.2 FLEXLM LICENSE MANAGER FOR WINDOWS

**lmgrd.exe** can be run manually or using the graphical Windows tool. You can start this tool from the FLEXlm program folder. Click on Start | Programs | TASKING FLEXlm | FLEXlm Tools



From the Control tab you can start, stop, and check the status of your license server. Select the Setup tab to enter information about your license server.



Select the **Control** tab and click the **Start** button to start your license server. **lmgrd.exe** will be launched as a background application with the license file and debug log file locations passed as parameters.

If you want **lmgrd.exe** to start automatically on NT, select the **Use NT Services** check box and **lmgrd.exe** will be installed as an NT service. Next, select the **Start Server at Power-UP** check box.

The **Licenses** tab provides information about the license file and the **Advanced** tab allows you to perform diagnostics and check versions.

## 4 THE DAEMON LOG FILE

The FLEXlm daemons all generate log files containing messages in the following format:

*mm/dd hh:mm (DAEMON name) message*

Where:

*mm/dd hh:mm* Is the month/day hour:minute that the message was logged.

*DAEMON name* Either “license daemon” or the string from the DAEMON line that describes your daemon.

In the case where a single copy of the daemon cannot handle all of the requested licenses, an optional “\_” followed by a number indicates that this message comes from a forked daemon.

*message* The text of the message.

The log files can be used to:

- Inform you when it may be necessary to update your application software licensing arrangement.
- Diagnose configuration problems.
- Diagnose daemon software errors.

The messages are grouped below into the above three categories, with each message followed by a brief description of its meaning.

## 4.1 INFORMATIONAL MESSAGES

### ***Connected to node***

This daemon is connected to its peer on node *node*.

### ***CONNECTED, master is name***

The license daemons log this message when a quorum is up and everyone has selected a master.

### ***DEMO mode supports only one SERVER host!***

An attempt was made to configure a demo version of the software for more than one server host.

### ***DENIED: N feature to user (mm/dd/yy hh:mm)***

*user* was denied access to *N* licenses of *feature*. This message may indicate a need to purchase more licenses.

### ***EXITING DUE TO SIGNAL mm***

### ***EXITING with code mm***

All daemons list the reason that the daemon has exited.

### ***EXPIRED: feature***

*feature* has passed its expiration date.

### ***IN: feature by user (N licenses) (used: d:hh:mm:ss) (mm/dd/yy hh:mm)***

*user* has checked back in *N* licenses of *feature* at *mm/dd/yy hh:mm*.

### ***IN server died: feature by user (number licenses) (used: d:hh:mm:ss) (mm/dd/yy hh:mm)***

*user* has checked in *N* licenses by virtue of the fact that his server died.

### ***License Manager server started***

The license daemon was started.

***Lost connection to host***

A daemon can no longer communicate with its peer on node *host*, which can cause the clients to have to reconnect, or cause the number of daemons to go below the minimum number, in which case clients may start exiting. If the license daemons lose the connection to the master, they will kill all the vendor daemons; vendor daemons will shut themselves down.

***Lost quorum***

The daemon lost quorum, so will process only connection requests from other daemons.

***MASTER SERVER died due to signal mm***

The license daemon received fatal signal *mm*.

***MULTIPLE xxx servers running. Please kill, and restart license daemon***

The license daemon has detected that multiple copies of vendor daemon *xxx* are running. The user should kill all *xxx* daemon processes and re-start the license daemon.

***OUT: feature by user (N licenses) (mm/dd/yy hh:mm)***

*user* has checked out *N* licenses of *feature* at *mm/dd/yy hh:mm*

***Removing clients of children***

The top-level daemon logs this message when one of the child daemons dies.

***RESERVE feature for HOST name******RESERVE feature for USER name***

A license of *feature* is reserved for either user *name* or host *name*.

***REStarted xxx (internet port mm)***

Vendor daemon *xxx* was restarted at internet port *mm*.

***Retrying socket bind (address in use)***

The license servers try to bind their sockets for approximately 6 minutes if they detect *address in use* errors.

***Selected (EXISTING) master node***

This license daemon has selected an existing master (node) as the master.

***SERVER shutdown requested***

A daemon was requested to shut down via a user-generated kill command.

***[NEW] Server started for: feature-list***

A (possibly new) server was started for the features listed.

***Shutting down xxx***

The license daemon is shutting down the vendor daemon *xxx*.

***SIGCHLD received. Killing child servers***

A vendor daemon logs this message when a shutdown was requested by the license daemon.

***Started name***

The license daemon logs this message whenever it starts a new vendor daemon.

***Trying connection to node***

The daemon is attempting a connection to *node*.

## **4.2 CONFIGURATION PROBLEM MESSAGES**

### ***hostname: Not a valid server host, exiting***

This daemon was run on an invalid hostname.

### ***hostname: Wrong hostid, exiting***

The hostid is wrong for *hostname*.

### ***BAD CODE for feature-name***

The specified feature name has a bad encryption code.

### ***CANNOT OPEN options file “file”***

The options file specified in the license file could not be opened.

### ***Couldn't find a master***

The daemons could not agree on a master.

### ***license daemon: lost all connections***

This message is logged when all the connections to a server are lost, which often indicates a network problem.

### ***lost lock, exiting***

#### ***Error closing lock file***

#### ***Unable to re-open lock file***

The vendor daemon has a problem with its lock file, usually because of an attempt to run more than one copy of the daemon on a single node. Locate the other daemon that is running via a **ps** command, and kill it with **kill -9**.

### ***NO DAEMON line for daemon***

The license file does not contain a DAEMON line for *daemon*.

### ***No “license” service found***

The TCP *license* service did not exist in `/etc/services`.

### ***No license data for “feat”, feature unsupported***

There is no feature line for *feat* in the license file.



***No features to serve!***

A vendor daemon found no features to serve. This could be caused by bad data in the license file.

***UNSUPPORTED FEATURE request: feature by user***

The *user* has requested a feature that this vendor daemon does not support. This can happen for a number of reasons: the license file is bad, the feature has expired, or the daemon is accessing the wrong license file.

***Unknown host: hostname***

The hostname specified on a `SERVER` line in the license file does not exist in the network database (probably `/etc/hosts`).

***lm\_server: lost all connections***

This message is logged when all the connections to a server are lost. This probably indicates a network problem.

***NO DAEMON lines, exiting***

The license daemon logs this message if there are no `DAEMON` lines in the license file. Since there are no vendor daemons to start, there is nothing to do.

***NO DAEMON line for name***

A vendor daemon logs this error if it cannot find its own `DAEMON` name in the license file.

### **4.3 DAEMON SOFTWARE ERROR MESSAGES**

#### ***accept: message***

An error was detected in the accept system call.

#### ***ATTEMPT TO START VENDOR DAEMON xxx with NO MASTER***

A vendor daemon was started with no master selected. This is an internal consistency error in the daemons.

#### ***BAD PID message from mm: pid: xxx (msg)***

A top-level vendor daemon received an invalid PID message from one of its children (daemon number xxx).

#### ***BAD SCONNECT message: (message)***

An invalid “server connect” message was received.

#### ***Cannot create pipes for server communication***

The pipe call failed.

#### ***Can't allocate server table space***

A malloc error. Check swap space.

#### ***Connection to node TIMED OUT***

The daemon could not connect to *node*.

#### ***Error sending PID to master server***

The vendor server could not send its PID to the top-level server in the hierarchy.

#### ***Illegal connection request to DAEMON***

A connection request was made to DAEMON, but this vendor daemon is not DAEMON.

#### ***Illegal server connection request***

A connection request came in from another server without a DAEMON name.

#### ***KILL of child failed, errno = mm***

A daemon could not kill its child.

***No internet port number specified***

A vendor daemon was started without an internet port.

***Not enough descriptors to re-create pipes***

The “top-level” daemon detected one of its sub-daemon’s death. In trying to restart the chain of sub-daemons, it was unable to get the file descriptors to set up the pipes to communicate. This is a fatal error, and the daemons must be re-started.

***read: error message***

An error in a read system call was detected.

***recycle\_control BUT WE DIDN'T HAVE CONTROL***

The hierarchy of vendor daemons has become confused over who holds the control token. This is an internal error.

***return\_reserved: can't find feature listhead***

When a daemon is returning a reservation to the “free reservation” list, it could not find the listhead of features.

***select: message***

An error in a select system call was detected.

***Server exiting***

The server is exiting. This is normally due to an error.

***SHELLO for wrong DAEMON***

This vendor daemon was sent a “server hello” message that was destined for a different DAEMON.

***Unsolicited msg from parent!***

Normally, the top-level vendor daemon sends no unsolicited messages. If one arrives, this message is logged. This is a bug.

***WARNING: CORRUPTED options list (o->next == 0)******Options list TERMINATED at bad entry***

An internal inconsistency was detected in the daemon’s option list.

## **5 FLEXLM LICENSE ERRORS**

### ***FLEXlm license error, encryption code in license file is inconsistent***

Check the contents of the license file using the license data sheet for the product. Correct the license file and run the **lmreread** command. However, do not change the last (fourth) field of a SERVER line in the license file. This cannot have any effect on the error message but changing it will cause other problems.

### ***license file does not support this version***

If this is a first time install then follow the procedure for the error message:

```
FLEXlm license error, encryption code in license file is
inconsistent
```

because there may be a typo in the fourth field of a FEATURE line of your license file. In all other cases you need a new license because the current license is for an older version of the product.

Replace the FEATURE line for the old version of the product with a FEATURE line for the new version (it can be found on the new license data sheet). Run the **lmreread** command afterwards. You can have only one version of a feature (previous versions of the product will continue to work).

### ***FLEXlm license error, cannot find license file***

Make sure the license file exists. If the pathname printed on the line after the error message is incorrect, correct this by setting the `LM_LICENSE_FILE` environment variable to the full pathname of the license file.

### ***FLEXlm license error, cannot read license file***

Every user needs to have read access on the license file and at least execute access on every directory component in the pathname of the license file. Write access is never needed. Read access on directories is recommended.

### ***FLEXlm license error, no such feature exists***

Check the license file. There should be a line starting with:

```
FEATURE SWiiiiii-jj
```

where "iiiiii" is a six digit software code and "jj" is a two digit host code for identifying a compatible host architecture. During product installations the product code is shown, e.g. SW008002, SW019002. The number in the software code is the same as the number in the product code except that the first number may contain an extra leading zero (it must be six digits long).

The line after the license error message describes the expected feature format and includes the host code.

Correct the license file using the license data sheet for the product and run the **lmreread** command. There is one catch: do not add extra SERVER lines or change existing SERVER lines in the license file.

### ***FLEXlm license error, license server does not support this feature***

If the LM\_LICENSE\_FILE variable has been set to the format *number@host* then see first the solution for the message:

```
FLEXlm license error, no such feature exists
```

Run the **lmreread** program to inform the license server about a changed license data file. If **lmreread** succeeds informing the license server but the error message persists, there are basically three possibilities:

1. The license key is incorrect. If this is the case then there must be an error message in the log file of **lmgrd**. Correct the key using the license data sheet for the product. Finally rerun **lmreread**. The log file of **lmgrd** is usually specified to **lmgrd** at startup with the **-l** option or with **>**.
2. Your network has more than one FLEXlm license server daemon and the default license file location for **lmreread** differs from the default assumed by the program. Also, there must be more than one license file. Try one of the following solutions on the same host which produced the error message:

- type:

```
lmreread -c /usr/local/flexlm/licenses/license.dat
```

- set LM\_LICENSE\_FILE to the license file location and retry the **lmreread** command.
- use the **lmreread** program supplied with the product SW000098, Flexible License Manager. SW000098 is bundled with all TASKING products.

3. There is a protocol version mismatch between **lmgrd** and the daemon with the name "Tasking" (the vendor daemon according to FLEXlm terminology) or there is some other internal error. These errors are always written to the log file of **lmgrd**. The solution is to upgrade the **lmgrd** daemon to the one supplied in SW000098, the bundled Flexible License Manager product.

On the other hand, if **lmreread** complains about not being able to connect to the license server then follow the procedure described in the next section for the error message "Cannot read license file data from server". The only difference with the current situation is that not the product but a license management utility shows a connect problem.

### ***FLEXlm license error, Cannot read license file data from server***

This indicates that the program could not connect to the license server daemon. This can have a number of causes. If the program did not immediately print the error message but waited for about 30 seconds (this can vary) then probably the license server host is down or unreachable. If the program responded immediately with the error message then check the following if the LM\_LICENSE\_FILE variable has been set to the format *number@host*:

- is the number correct? It should match the fourth field of a SERVER line in the license file on the license server host. Also, the host name on that SERVER line should be the same as the host name set in the LM\_LICENSE\_FILE variable. Correct LM\_LICENSE\_FILE if necessary.

In any case one should verify if the license server daemon is running. Type the following command on the host where the license server daemon (**lmgrd**) is supposed to run.

On SunOS 4.x:

```
ps wwax | grep lmgrd | grep -v grep
```

On HP-UX or SunOS 5.x (Solaris 2.x):

```
ps -ef | grep lmgrd | grep -v grep
```

If the command does not produce any output then the license server daemon is not running. See below for an example how to start **lmgrd**.

Make sure that both license server daemon (**lmgrd**) and the program are using the same license data. All TASKING products use the license file `/usr/local/flexlm/licenses/license.dat` unless overruled by the environment variable `LM_LICENSE_FILE`. However, not all existing **lmgrd** daemons may use the same default. In case of doubt, specify the license file pathname with the `-c` option when starting the license server daemon. For example:

```
lmgrd -c /usr/local/flexlm/licenses/license.dat \
-l /usr/local/flexlm/licenses/license.log &
```

and set the `LM_LICENSE_FILE` environment variable to the `license.dat` pathname mentioned with the `-c` option of **lmgrd** before running any license based program (including **lmreread**, **lmstat**, **lmdown**). If **lmgrd** and the program run on different hosts, transparent access to the license file is assumed in the situation described above (e.g. NFS). If this is not the case, make a local copy of the license file (not recommended) or set `LM_LICENSE_FILE` to the form *number@host*, as described earlier.

If none of the above seems to apply (i.e. **lmgrd** was already running and `LM_LICENSE_FILE` has been set correctly) then it is very likely that there is a TCP port mismatch. The fourth field of a `SERVER` line in the license file specifies a TCP port number. That number can be changed without affecting any license. However, it must never be changed while the license server daemon is running. If it has been changed, change it back to the original value. If you do not know the original number anymore, restart the license server daemon after typing the following command on the license server host:

```
kill PID
```

where `PID` is the process id of **lmgrd**.

## **6 FREQUENTLY ASKED QUESTIONS (FAQS)**

### **6.1 LICENSE FILE QUESTIONS**

***I've received FLEXlm license files from 2 different companies. Do I have to combine them?***

You don't have to combine license files. Each license file that has any 'counted' lines (the 'number of licenses' field is >0) requires a server. It's perfectly OK to have any number of separate license files, with different **lmgrd** server processes supporting each file. Moreover, since **lmgrd** is a lightweight process, for sites without system administrators, this is often the simplest (and therefore recommended) way to proceed. With v6+ **lmgrd/lmdown/lmreread**, you can stop/reread/restart a single vendor daemon (of any FLEXlm version). This makes combining licenses more attractive than previously. Also, if the application is v6+, using 'dir/\*.lic' for license file management behaves like combining licenses without physically combining them.

***When is it recommended to combine license files?***

Many system administrators, especially for larger sites, prefer to combine license files to ease administration of FLEXlm licenses. It's purely a matter of preference.

***Does FLEXlm handle dates in the year 2000 and beyond?***

Yes. The FLEXlm date format uses a 4-digit year. Dates in the 20th century (19xx) can be abbreviated to the last 2 digits of the year (xx), and use of this feature is quite widespread. Dates in the year 2000 and beyond must specify all 4 year digits.

### **6.2 FLEXLM VERSION**

***Which FLEXlm versions does TASKING deliver?***

For Windows we deliver FLEXlm v6.1 and for UNIX we deliver v2.4.



***I have products from several companies at various FLEXlm version levels. Do I have to worry about how these versions work together?***

If you're not combining license files from different vendors, the simplest thing to do is make sure you use the tools (especially **lmgrd**) that are shipped by each vendor.

**lmgrd** will always correctly support older versions of vendor daemons and applications, so it's **always** safe to use the latest version of **lmgrd** and the other FLEXlm utilities. If you've combined license files from 2 vendors, you **must** use the latest version of **lmgrd**.

If you've received 2 versions of a product from the same vendor, you must use the latest vendor daemon they sent you. An older vendor daemon with a newer client will cause communication errors.

Please ignore letters appended to FLEXlm versions, i.e., v2.4d. The appended letter indicates a patch, and does NOT indicate any compatibility differences. In particular, some elements of FLEXlm didn't require certain patches, so a 2.4 **lmgrd** will work successfully with a 2.4b vendor daemon.

***I've received a new copy of a product from a vendor, and it uses a new version of FLEXlm. Is my old license file still valid?***

Yes. Older FLEXlm license files are always valid with newer versions of FLEXlm.

## **6.3 WINDOWS QUESTIONS**

***What Windows Host Platforms can be used as a server for Floating Licenses?***

The system being used as the server (where the FLEXlm License Manager is running) for Floating licenses, must be Windows NT. The FLEXlm License Manager does not run properly with Windows 95/98.

***Why do I need to include NWlink IPX/SPX on NT?***

This is necessary for either obtaining the Ethernet card address, or to provide connectivity with a Netware License server.

## 6.4 TASKING QUESTIONS

### *How will the TASKING licensing/pricing model change with License Management (FLEXlm)?*

TASKING will now offer the following types of licenses so you can purchase licenses based upon usage:

| License     | Description                                                                                                                                                                                                    | Pricing                                                                       |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| Node Locked | This license can only be used on a specific system. It cannot be moved to another system.                                                                                                                      | The pricing for this license will be the current product pricing.             |
| Floating    | This license requires a network (license server and a TCP/IP (or IPX/SPX) connection between clients and server) and can be used on any host system ( <b>using the same operating system</b> ) in the network. | The pricing for this license will be 50% higher than the node locked license. |

### *How does FLEXlm affect future product ordering?*

For all licenses, node locked or floating, you must provide information that is used to create a license key. For node locked licenses we must have the HOST ID. Floating licenses require the HOST ID and HOST NAME. The HOST ID is a unique identification of the machine, which is based upon different hardware depending upon host platform. The HOST NAME is the network name of the machine.



TASKING Logistics CANNOT ship ANY orders that do not include the HOST ID and/or HOST NAME information.

### *What if I do not know the information needed for the license key?*

We have a software utility (**tkhostid.exe**) which will obtain and display the HOST ID so a customer can easily obtain this information. This utility is available from our web site, placed on all product CDs (which support FLEXlm), and from technical support. If you have already installed FLEXlm, you can also use **lmhostid**.

- In the case of a *Node locked license*, it is important that the customer runs this utility on the exact machine he intends to run the TASKING tools on.

- In the case of a *Floating License*, the **tkhostid.exe** (or **lmhostid**) utility should be run on the machine on which the FLEXlm license manager will be installed, e.g. the server. The HOST NAME information can be obtained from within the Windows Control Panel. Select "Network", click on "Identification", look for "Computer name".

### ***How will the "locking" mechanism work?***

- For node locked licenses, FLEXlm will first search for an ethernet card. If one exists, it will lock onto the number of the ethernet card. If an ethernet card does not exist, FLEXlm will lock onto the hard disk serial number.
- For floating licenses, the ethernet card number will be used.

### ***What happens if I try to move my node locked license to another system?***

The software will not run.

### ***What does linger-time for floating licenses mean?***

When the TASKING product starts to run, it will try to obtain a license from the license server. The license server keeps track of the number of licenses already issued, and grants or denies the request. When the software has finished running, the license is kept by the license server for a period of time known as the "linger-time". If the same user requests the TASKING product again within the linger-time, he is granted the license again. If another user requests a license during the linger-time, his request is denied until the linger-time has finished.

### ***What is the length of the linger-time for floating licenses?***

The length of the linger-time for both the PC and UNIX floating licenses is 5 minutes.

### ***Can the linger-time be changed?***

Yes. A customer can change the linger-time to be larger (but not shorter) than the time specified by TASKING.

### ***What happens if my system crashes or I upgrade to a new system?***

You will need to contact Technical Support for temporary license keys due to a system crash or to move from one system to another system. You will then need to work with your local sales representative to obtain a permanent new license key.

## 6.5 USING FLEXLM FOR FLOATING LICENSES

### ***Does FLEXlm work across the internet?***

Yes. A server on the internet will serve licenses to anyone else on the internet. This can be limited with the 'INTERNET=' attribute on the FEATURE line, which limits access to a range of internet addresses. You can also use the INCLUDE and EXCLUDE options in the daemon option file to allow (or deny) access to clients running on a range of internet addresses.

### ***Does FLEXlm work with Internet firewalls?***

Many firewalls require that port numbers be specified to the firewall. FLEXlm v5 **lmgrd** supports this.

### ***If my client dies, does the server free the license?***

Yes, unless the client's whole system crashes. Assuming communications is TCP, the license is automatically freed immediately. If communications are UDP, then the license is freed after the UDP timeout, which is set by each vendor, but defaults to 45 minutes. UDP communications is normally only set by the end-user, so TCP should be assumed. If the whole system crashes, then the license is not freed, and you should use '**lmremove**' to free the license.

### ***What happens when the license server dies?***

FLEXlm applications send periodic heartbeats to the server to discover if it has died. What happens when the server dies is then up to the application. Some will simply continue periodically attempting to re-checkout the license when the server comes back up. Some will attempt to re-checkout a license a few times, and then, presumably with some warning, exit. Some GUI applications will present pop-ups to the user periodically letting them know the server is down and needs to be re-started.

### ***How do you tell if a port is already in use?***

99.44% of the time, if it's in use, it's because **lmgrd** is already running on the port – or was recently killed, and the port isn't freed yet. Assuming this is not the case, then use '**telnet host port**' – if it says "*can't connect*", it's a free port.

### ***Does FLEXlm require root permissions?***

No. There is no part of FLEXlm, **lmgrd**, vendor daemon or application, that requires root permissions. In fact, it is strongly recommended that you do not run the license server (**lmgrd**) as root, since root processes can introduce security risks.

If **lmgrd** must be started from the root user (for example, in a system boot script), we recommend that you use the '**su**' command to run **lmgrd** as a non-privileged user:

```
su username -c"/path/lmgrd -c /path/license.dat \
-l /path/log"
```

where *username* is a non-privileged user, and *path* is the correct paths to **lmgrd**, *license.dat* and debug log file. You will have to ensure that the vendor daemons listed in */path-to-license/license.dat* have execute permissions for *username*. The paths to all the vendor daemons in the license file are listed on each DAEMON line.

### ***Is it ok to run lmgrd as 'root' (UNIX only)?***

It is not prudent to run any command, particularly a daemon, as root on UNIX, as it may pose a security risk to the Operating System. Therefore, we recommend that **lmgrd** be run as a non-privileged user (not 'root'). If you are starting **lmgrd** from a boot script, we recommend that you use

```
su username -c"umask 022; /path/lmgrd \
-c /path/license.dat -l /path/log"
```

to run **lmgrd** as a non-privileged user.

### ***Does FLEXlm licensing impose a heavy load on the network?***

No, but partly this depends on the application, and end-user's use. A typical checkout request requires 5 messages and responses between client and server, and each message is < 150 bytes.

When a server is not receiving requests, it requires virtually no CPU time. When an application, or **lmstat**, requests the list of current users, this can significantly increase the amount of networking FLEXlm uses, depending on the number of current users. Also, prior to FLEXlm v5, use of 'port@host' can increase network load, since the license file is down-loaded from the server to the client. 'port@host' should be, if possible, limited to small license files (say < 50 features). In v5, 'port@host' actually improves performance.

***Does FLEXlm work with NFS?***

Yes. FLEXlm has no direct interaction with NFS. FLEXlm uses an NFS-mounted file like any other application.

***Does FLEXlm work with ATM, ISDN, Token-Ring, etc.?***

In general, these have no impact on FLEXlm. FLEXlm requires TCP/IP or SPX (Novell Netware). So long as TCP/IP works, FLEXlm will work.

***Does FLEXlm work with subnets, fully-qualified names, multiple domains, etc.?***

Yes, although this behavior was improved in v3.0, and v6.0. When a license server and a client are located in different domains, fully-qualified host names have to be used. A fully-qualified hostname is of the form:

*node.domain*

where *node* is the local hostname (usually returned by the '**hostname**' command or '**uname -n**') *domain* is the internet domain name, e.g. 'globes.com'.

To ensure success with FLEXlm across domains, do the following:

1. Make the sure the fully-qualified hostname is the name on the SERVER line of the license file.
2. Make sure ALL client nodes, as well as the server node, are able to 'telnet' to that fully-qualified hostname. For example, if the host is locally called 'speedy', and the domain name is 'corp.com', local systems will be able to logon to speedy via 'telnet speedy'. But very often, 'telnet speedy.corp.com' will fail, locally.  
Note that this telnet command will always succeed on hosts in other domains (assuming everything is configured correctly), since the network will resolve speedy.corp.com automatically.
3. Finally, there must be an 'alias' for speedy so it's also known locally as speedy.corp.com. This alias is added to the `/etc/hosts` file, or if NIS/Yellow Pages are being used, then it will have to be added to the NIS database. This requirement goes away in version 3.0 of FLEXlm.

If all components (application, **lmgrd** and vendor daemon) are v6.0 or higher, no aliases are required; the only requirement is that the fully-qualified domain name, or IP-address, is used as a hostname on the SERVER, or as a hostname in `LM_LICENSE_FILE port@host, or @host`.

### ***Does FLEXlm work with NIS and DNS?***

Yes. However, some sites have broken NIS or DNS, which will cause FLEXlm to fail. In v5 of FLEXlm, NIS and DNS can be avoided to solve this problem. In particular, sometimes DNS is configured for a server that's not current available (e.g., a dial-up connection from a PC). Again, if DNS is configured, but the server is not available, FLEXlm will fail.

In addition, some systems, particularly Sun, SGI, HP, require that applications be linked dynamically to support NIS or DNS. If a vendor links statically, this can cause the application to fail at a site that uses NIS or DNS. In these situations, the vendor will have to relink, or recompile with v5 FLEXlm. Vendors are strongly encouraged to use dynamic libraries for libc and networking libraries, since this tends to improve quality in general, as well as making NIS/DNS work.

On PCs, if a checkout seems to take 3 minutes and then fails, this is usually because the system is configured for a dial-up DNS server which is not currently available. The solution here is to turn off DNS.

Finally, hostnames must NOT have periods in the name. These are not legal hostnames, although PCs will allow you to enter them, and they will not work with DNS.

### ***We're using FLEXlm over a wide-area network. What can we do to improve performance?***

FLEXlm network traffic should be minimized. With the most common uses of FLEXlm, traffic is negligible. In particular, checkout, checkin and heartbeats use very little networking traffic. There are two items, however, which can send considerably more data and should be avoided or used sparingly:

- **'lmstat -a'** should be used sparingly. **'lmstat -a'** should not be used more than, say, once every 15 minutes, and should be particularly avoided when there's a lot of features, or concurrent users, and therefore a lot of data to transmit; say, more than 20 concurrent users or features.
- Prior to FLEXlm v5, the 'port@host' mode of the LM\_LICENSE\_FILE environment variable should be avoided, especially when the license file has many features, or there are a lot of license files included in LM\_LICENSE\_FILE. The license file information is sent via the network, and can place a heavy load. Failures due to 'port@host' will generate the error LM\_SERVNOREADLIC (-61).

# APPENDIX

## B

### MISRA C

---





---

**B**

**APPENDIX**

---

***Supported and unsupported MISRA C rules***

1. no language extensions shall be used
- \* 2. other languages should only be used with an interface standard
3. inline assembly is only allowed in dedicated C functions
- \* 4. provision should be made for appropriate run-time checking
5. only use characters defined by the C standard
- \* 6. character values shall be restricted to a subset of ISO 106460-1
7. trigraphs shall not be used
8. multibyte characters and wide string literals shall not be used
9. comments shall not be nested
- \* 10. sections of code should not be "commented out"
11. identifiers shall not rely on significance of more than 31 characters
12. the same identifier shall not be used in multiple name spaces
13. specific-length typedefs should be used instead of the basic types
14. use 'unsigned char' or 'signed char' instead of plain 'char'
- \* 15. floating point implementations should comply with a standard
- \* 16. the bit representation of floating point numbers shall not be used
17. typedef names should not be reused
- \* 18. numeric constants should be suffixed to indicate type
19. octal constants (other than zero) shall not be used
20. all object and function identifiers shall be declared before use
21. identifiers shall not hide identifiers in an outer scope
22. declarations should be at function scope where possible ("static variable")
- \* 23. all declarations at file scope should be static where possible
24. identifiers shall not have both internal and external linkage

- \* 25. identifiers with external linkage shall have exactly one definition
- 26. multiple declarations for objects or functions shall be compatible
- \* 27. external objects should not be declared in more than one file
- 28. the 'register' storage class specifier should not be used
- 29. the use of a tag shall agree with its declaration
- 30. all automatics shall be initialized before being used
- 31. braces shall be used in the initialization of arrays and structures
- 32. only the first, or all enumeration constants may be initialized
- 33. the right hand side of && or || shall not contain side effects
- 34. the operands of a logical && or || shall be primary expressions
- 35. assignment operators shall not be used in Boolean expressions
- \* 36. logical operators should not be confused with bitwise operators
- 37. bitwise operations shall not be performed on signed integers
- 38. a shift count shall be between 0 and the operand width minus 1
- 39. the unary minus shall not be applied to an unsigned expression
- 40. 'sizeof' should not be used on expressions with side effects
- \* 41. the implementation of integer division should be documented
- 42. the comma operator shall only be used in a 'for' condition
- 43. don't use implicit conversions which may result in information loss
- 44. redundant explicit casts should not be used
- 45. type casting from any type to/from pointers shall not be used
- 46. the value of an expression shall be evaluation order independent
- \* 47. no dependence should be placed on operator precedence rules
- \* 48. mixed arithmetic should use explicit casting
- \* 49. tests of a (non-Boolean) value against 0 should be made explicit

- 50. F.P. variables shall not be tested for exact equality or inequality
- \* 51. constant unsigned integer expressions should not wrap-around
- 52. there shall be no unreachable code
- 53. all non-null statements shall have a side-effect
- 54. a null statement shall only occur on a line by itself
- 55. labels should not be used
- 56. the 'goto' statement shall not be used
- 57. the 'continue' statement shall not be used
- 58. the 'break' statement shall not be used (except in a 'switch')
- 59. an 'if' or loop body shall always be enclosed in braces
- 60. all 'if', 'else if' constructs should contain a final 'else'
- 61. every non-empty 'case' clause shall be terminated with a 'break'
- 62. all 'switch' statements should contain a final 'default' case
- 63. a 'switch' expression should not represent a Boolean case
- 64. every 'switch' shall have at least one 'case'
- 65. floating point variables shall not be used as loop counters
- \* 66. a "for" should only contain expressions concerning loop control
- \* 67. iterator variables should not be modified in a "for" loop
- 68. functions shall always be declared at file scope
- 69. functions with variable number of arguments shall not be used
- 70. functions shall not call themselves
- 71. function prototypes shall be visible at the definition and call
- 72. the function prototype of the declaration shall match the definition
- 73. identifiers shall be given for all prototype parameters or for none
- 74. parameter identifiers shall be identical for declaration/definition

- 75. every function shall have an explicit return type
- 76. functions with no parameters shall have a 'void' parameter list
- \* 77. an actual parameter type shall be compatible with the prototype
- 78. the number of actual parameters shall match the prototype
- 79. the values returned by 'void' functions shall not be used
- 80. void expressions shall not be passed as function parameters
- \* 81. "const" should be used for reference parameters not modified
- 82. a function should have a single point of exit
- 83. every exit point shall have a 'return' of the declared return type
- 84. for 'void' functions, 'return' shall not have an expression
- 85. function calls with no parameters should have empty parentheses
- \* 86. if a function returns error information, it should be tested
- 87. #include shall only be preceded by another directives or comments
- 88. non-standard characters shall not occur in #include directives
- 89. #include shall be followed by either <filename> or "filename"
- 90. plain macros shall only be used for constants/qualifiers/specifiers
- 91. macros shall not be defined/undefined within a block
- 92. #undef should not be used
- \* 93. a function should be used in preference to a function-like macro
- 94. a function-like macro shall not be used without all arguments
- \* 95. macro arguments shall not contain pre-preprocessing directives
- 96. macro definitions/parameters should be enclosed in parentheses
- 97. don't use undefined identifiers in pre-processing directives
- 98. a macro definition shall contain at most one # or ## operator

- \* 99. all uses of the `#pragma` directive shall be documented
- 100. `'defined'` shall only be used in one of the two standard forms
- 101. pointer arithmetic should not be used
- 102. no more than 2 levels of pointer indirection should be used
- \* 103. no relational operators between pointers to different objects
- 104. non-constant pointers to functions shall not be used
- 105. functions assigned to the same pointer shall be of identical type
- 106. an automatic address may not be assigned to a longer lived object
- \* 107. the null pointer shall not be de-referenced
- \* 108. all struct/union members shall be fully specified
- \* 109. overlapping variable storage shall not be used
- \* 110. unions shall not be used to access the sub-parts of larger types
- 111. bit fields shall have type `'unsigned int'` or `'signed int'`
- 112. bit fields of type `'signed int'` shall be at least 2 bits long
- 113. all struct/union members shall be named
- 114. reserved and standard library names shall not be redefined
- 115. standard library function names shall not be reused
- \* 116. production libraries shall comply with the MISRA C restrictions
- \* 117. the validity of library function parameters shall be checked
- 118. dynamic heap memory allocation shall not be used
- 119. `'errno'` should not be used
- 120. the macro `'offsetof()'` shall not be used
- 121. `<locale.h>` and the `'setlocale'` function shall not be used
- 122. the `'setjmp'` and `'longjmp'` functions shall not be used
- 123. the signal handling facilities of `<signal.h>` shall not be used

- 124. the <stdio.h> library shall not be used in production code
- 125. the functions atof/atoi/atol shall not be used
- 126. the functions abort/exit/getenv/system shall not be used
- 127. the time handling functions of library <time.h> shall not be used



\* = Not supported by the TASKING TriCore C compiler

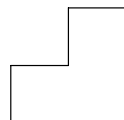
# APPENDIX C

---

## **CPU FUNCTIONAL PROBLEMS**



**TASKING**





---

**C**

**APPENDIX**

---

## 1 INTRODUCTION

Infineon Technologies regularly publishes microcontroller errata sheets reporting functional problems and deviations from the electrical specifications and timing specifications.

The TASKING TriCore software development tools provide solutions for a number of these functional problems in the TriCore architecture.

Support to deal with CPU functional problem is provided in three areas:

- Whenever possible and relevant, compiler bypasses will modify the code in order to avoid the identified erroneous code sequences;
- The TriCore assembler gives warnings for suspicious or erroneous code sequences;
- Ready-build, 'protected' standard C libraries with bypasses for all identified TriCore CPU functional problems are included in the tool chain.

This appendix lists a summary of identified functional problems which can be bypassed by the TASKING TriCore tool kit.

Please refer to the Infineon errata sheets for the TriCore architecture revision-step of your particular device, to check the need for applying any of these bypasses. Also refer to the Infineon errata sheets for a complete description of the CPU functional problems, as the workarounds listed below do not describe the functional problem itself.

The syntax used by Infineon to identify a CPU functional problems is:

*TC<architecture\_nr><version>\_<module\_name><problem\_nr>*

For example: **TC113\_CPU5** (TC1, version 1.3, module "CPU", problem #5)

With the TASKING C compiler and assembler command line options, pragmas and macro definitions you can enable or disable specific CPU functional problem bypasses.

To enable the compiler bypasses and assembler checks for *all* TriCore CPU TC112 problems (respectively TC113 problems) at once, use the command line option **-zTC112\_DEFECTS** or **#pragma TC112\_DEFECTS** (respectively **-zTC113\_DEFECTS** or **#pragma TC113\_DEFECTS**)

In the embedded development environment (EDE) you can enable as follows:



Select the EDE | Processor Options... menu and choose the Bypasses TC1 v1.2 tab or the Bypasses TC1 v1.3 tab. Then select the bypasses you want to enable.

The table below shows an overview of all CPU functional problems.

| TC Version | Functional Problem |
|------------|--------------------|
| 112        | COR1               |
| 112        | COR3               |
| 112        | COR4               |
| 112        | COR6               |
| 112        | COR10              |
| 112        | COR13              |
| 112        | COR14              |
| 112        | COR15              |
| 112        | COR16              |
| 112        | COR17              |
| 113        | PMU1               |
| 113        | PMU3               |
| 113        | CPU5               |
| 113        | CPU9               |
| 113        | CPU11              |
| 113        | CPU13              |
| 113        | CPU14              |
| 113        | CPU15              |
| 113        | CPU16              |
| 113        | DMU1               |
| 113        | LFI2               |
| 113        | LFI3               |

Table C-1: Overview of supported TriCore CPU functional problems

## **2 CPU FUNCTIONAL PROBLEM BYPASSES TC1 V1.2**

### **2.1 TC112\_COR1**

#### **Compiler and assembler option:**

**-zTC112\_COR1**

#### **Pragma:**

**#pragma TC112\_COR1 [on | off | restore]**

#### **Assembler control:**

**\$TC112\_COR1 {on | off}**

#### **Assembler macro:**

The assembler macro `_TC112_COR1` is defined if you specify the option **-zTC112\_COR1**.

#### **Protected libraries to link:**

`lib\p\tc112\*.a`

#### **Compiler bypass:**

To bypass this CPU functional problem, the C compiler generates an ISYNC instruction before each LOOP, LOOP16 and LOOPU instruction.

#### **Assembler check:**

The assembler gives a warning when the preceding instruction of a LOOP, LOOP16 or LOOPU instruction is not an ISYNC instruction:

```
171 suspicious instruction concerning CPU functional
defect TC112_COR1
```

You can suppress this warning with the option **-w171**.

## **2.2 TC112\_COR3**

### **Locator option:**

`-em_TC112_COR3`

To bypass this CPU functional problem, a locator control is used in the `tri.cpu` descriptor to restrict the size in the CSA absolute address mapping to 32Kb scratch pad RAM on the DMU.

## **2.3 TC112\_COR4**

### **Compiler and assembler option:**

**-zTC112\_COR4**

### **Pragma:**

**#pragma TC112\_COR4 [on | off | restore]**

### **Assembler control:**

**\$TC112\_COR4 {on | off}**

### **Assembler macro:**

The assembler macro `_TC112_COR4` is defined if you specify the option **-zTC112\_COR4**.

### **Protected libraries to link:**

`lib\p\tc112\*.a`

### **Compiler bypass:**

To bypass this CPU functional problem, the C compiler generates a NOP instruction between a (target) label and the instruction following it. This is done when the instruction directly uses an `An` register for either an effective address calculation or as the target of an indirect branch.

### **Assembler check:**

The assembler gives a warning for an instruction using an `An` register for either an effective address calculation or as the target of an indirect branch that is located directly after a (target) label:

```
172 suspicious instruction concerning CPU functional
defect TC112_COR4
```

You can suppress this warning with the option **-w172**.

## **2.4 TC112\_COR6**

### **Assembler option:**

**-zTC112\_COR6**

### **Assembler control:**

**\$TC112\_COR6 {on | off}**

### **Assembler macro:**

The assembler macro `_TC112_COR6` is defined if you specify the option **-zTC112\_COR6**.

### **Protected libraries to link:**

`lib\p\tc112\*.a`

### **Compiler bypass:**

There is no C compiler workaround required for this CPU functional problem, because the compiler does not generate `CALLI` instructions with a target address in register A11.

### **Assembler check:**

The assembler generates an error for instruction `CALLI A11`.

## **2.5 TC112\_COR10**

### **Compiler and assembler option:**

**-zTC112\_COR10**

### **Pragma:**

**#pragma TC112\_COR10 [on | off | restore]**

### **Assembler control:**

**\$TC112\_COR10 {on | off}**

### **Assembler macro:**

The assembler macro `_TC112_COR10` is defined if you specify the option **-zTC112\_COR10**.

### **Protected libraries to link:**

`lib\p\tc112\*.a`

### **Compiler bypass:**

To bypass this CPU functional problem, the C compiler avoids generation of store instructions that use a circular addressing mode with an offset value not equal to zero. An additional circular load instruction is generated with the required offset to post-increment the circular buffer pointer.

For example:

```
st.w [a6/a7+c]0,d15
ld.w d15,[a6/a7+c]4
```

Instead of:

```
st.w [a6/a7+c]4,d15
```

### **Assembler check:**

The assembler gives a warning for store operations that use a circular addressing mode with an offset not equal to zero:

```
173 suspicious instruction concerning CPU functional
defect TC112_COR10
```

You can suppress this warning with the option **-w173**.



## **2.6 TC112\_COR13**

### **Compiler and assembler option:**

**-zTC112\_COR13**

### **Pragma:**

**#pragma TC112\_COR13 [on | off | restore]**

### **Assembler control:**

**\$TC112\_COR13 {on | off}**

### **Assembler macro:**

The assembler macro `_TC112_COR13` is defined if you specify option **-zTC112\_COR13**.

### **Protected libraries to link:**

`lib\p\tc112\*.a`

### **Compiler bypass:**

To bypass this CPU functional problem, the C compiler generates a NOP prior to the LOOP instruction if the loop contains a single integer instruction that is a DVSTEP or a DVSTEP.U.

### **Assembler check:**

The assembler gives a warning for loops that contain a single integer instruction that is a DVSTEP or a DVSTEP.U:

```
174 suspicious instruction concerning CPU functional
defect TC112_COR13
```

You can suppress this warning with the option **-w174**.

## **2.7 TC112\_COR14**

### **Compiler and assembler option:**

**-zTC112\_COR14**

### **Pragma:**

**#pragma TC112\_COR14 [on | off | restore]**

### **Assembler control:**

**\$TC112\_COR14 {on | off}**

### **Assembler macro:**

The assembler macro `_TC112_COR14` is defined if you specify the option **-zTC112\_COR14**.

### **Protected libraries to link:**

`lib\p\tc112\*.a`

### **Compiler bypass:**

To bypass this CPU functional problem, the C compiler uses code that protects a divide instruction sequence against interrupts. Instead of generating inline divide code, the C compiler generates calls to run-time library functions that support divide operations with interrupt protection. Next skeleton code demonstrates the protective code used in these run-time library functions:

```
;;
;; Save interrupt state and disable interrupts
;;
mfcrr d0,#0xfe2c ; save ICR in d0
disable ; disable interrupts
```

divide instructions:

```
;;
;; Restore interrupt state
;;
jz.t d0:8,disabled ; do not enable interrupts
enable ; when they were disabled
disabled:
```

The C run-time library modules involved are `acircint.asm`, `dfrfr.asm`, `sdivmod.asm` and `udivmod.asm`.

**Assembler check:**

An assembler check for this CPU functional problem is not available, because global interrupt enable state can not be checked at assembly level.

## **2.8 TC112\_COR15**

### **Assembler option:**

**-zTC112\_COR15**

### **Assembler control:**

**\$TC112\_COR15 {on | off}**

### **Assembler macro:**

The assembler macro `_TC112_COR15` is if you specify the option **-zTC112\_COR15**.

### **Protected libraries to link:**

`lib\p\tc112\*.a` (or add `lib\src\cstart.asm` to your project).

### **Compiler bypass:**

To bypass this CPU functional problem, an assembler macro is added to the C startup code to disable the starvation protection by resetting the `BCUCON.SPE` bit.

### **Assembler check:**

No assembler check is supported, because run time checking of starvation protection is not possible at assembly level.

## **2.9 TC112\_COR16**

### **Compiler and assembler option:**

**-zTC112\_COR16**

### **Linker option:**

**-em\_TC112\_COR16**

### **Pragma:**

**#pragma TC112\_COR16 [on | off | restore]**

### **Assembler control:**

**\$TC112\_COR16 {on | off}**

### **Assembler macro:**

The assembler macro `_TC112_COR16` is defined if you specify the option **-zTC112\_COR16**.

### **Protected libraries to link:**

`lib\p\tc112\*.a` (or add `lib\src\cstart.asm` to your project).

### **Compiler bypass:**

To bypass this CPU functional problem, the C compiler aligns circular qualified buffers to a quad-word boundary, and the compiler sizes all stack frames to an integral number of quad-words. An assembler macro is added to the C startup code to enable initialization of the stack pointers to a quad-word boundary. A locator control is used in the `tri.dsc` descriptor to set the alignment of the user stack and the interrupt stack to a quad-word alignment. See section 3.20, *Circular Buffers* for a description on how to declare a circular buffer.

### **Assembler check:**

No assembler check is supported, because circular storage type info is not available at assembly level.

## **2.10 TC112\_COR17**

### **Compiler and assembler option:**

**-zTC112\_COR17**

### **Pragma:**

**#pragma TC112\_COR17 [on | off | restore]**

### **Assembler control:**

**\$TC112\_COR17 {on | off}**

### **Assembler macro:**

The assembler macro `_TC112_COR17` is defined if you specify the option **-zTC112\_COR17**.

### **Protected libraries to link:**

`lib\p\tc112\*.a`

### **Compiler bypass:**

To bypass this CPU functional problem, the C compiler generates a NOP instruction after a DSYNC instruction. The C compiler only generates a DSYNC instruction when bypass TC113\_CPU17 is enabled.

### **Assembler check:**

The assembler gives a warning if a DSYNC is not followed by a NOP instruction:

```
175 suspicious instruction concerning CPU functional
defect TC112_COR17
```

You can suppress this warning with the option **-w175**.

## **3 CPU FUNCTIONAL PROBLEM BYPASSES TC1 V1.3**

### **3.1 TC113\_PMU1**

#### **Assembler option:**

`-zTC113_PMU1`

#### **Protected libraries to link:**

`lib\p\tc113\*.a`, or add `lib\src\cstart.asm` to your project.

#### **Assembler macro:**

The assembler macro `_TC113_PMU1` is defined if you specify the option **`-zTC113_PMU1`**.

#### **Compiler bypass:**

To bypass this CPU functional problem, an assembler macro is added to the C startup code to disable the split mode on the LMB bus. The SPLT bit of the SFR register `LFI_CON` is set to zero.

#### **Assembler check:**

No assembler check is supported, because run time split mode can not be checked at assembly level.

## **3.2 TC113\_PMU3**

### **Assembler option:**

**-zTC113\_PMU3**

### **Assembler macro:**

The assembler macro `_TC113_PMU3` is defined if you specify the option **-zTC113\_PMU3**.

### **Protected libraries to link:**

`lib\p\tc113\*.a` (or add `lib\src\cstart.asm` to your project).

### **Compiler bypass:**

To bypass this CPU functional problem, an assembler macro is added to the C startup code to set the TLB-A and TLB-B mappings to a page size of 16 Kb. The SZA and SZB in the MMU\_CON are set to 16 Kb.

### **Assembler check:**

No assembler check is supported, because run time TBL mappings can not be checked at assembly level.



### 3.3 TC113\_CPU5

#### Compiler and assembler option:

**-zTC113\_CPU5**

#### Pragma:

**#pragma TC113\_CPU5 [on | off | restore]**

#### Assembler control:

**\$TC113\_CPU5 {on | off}**

#### Assembler macro:

The assembler macro `_TC113_CPU5` is defined if you specify the option **-zTC113\_CPU5**.

#### Protected libraries to link:

`lib\p\tc113\*.a`

#### Compiler bypass:

To bypass this CPU functional problem, the C compiler generates an ISYNC instruction before a loop body.

Example:

```

 isync
_loop_start:
 ..
 ..
 loop a8, _loop_start

```

#### Assembler check:

This CPU functional problem does not cause a run-time problem, it is only a performance issue. Therefore no assembler checking is required to warn you for possible run-time problems.

### **3.4 TC113\_CPU9**

**Compiler and assembler option:**

**-zTC113\_CPU9**

**Pragma:**

**#pragma TC113\_CPU9 [on | off | restore]**

**Assembler control:**

**\$TC113\_CPU9 {on | off}**

**Assembler macro:**

The assembler macro `_TC113_CPU9` is defined if you specify the option **-zTC113\_CPU9**.

**Protected libraries to link:**

`lib\p\tc113\*.a`

**Compiler bypass:**

To bypass this CPU functional problem, the C compiler generates two NOP instructions after a DSYNC instruction. The C compiler only generates a DSYNC instruction when CPU functional problem bypass TC113\_CPU14 is enabled.

**Assembler check:**

The assembler gives a warning if a DSYNC is not followed by two NOP instructions:

```
176 suspicious instruction concerning CPU functional
defect TC113_CPU9
```

You can suppress this warning with the option **-w176**.

### **3.5 TC113\_CPU11**

#### **Compiler and assembler option:**

**-zTC113\_CPU11**

#### **Pragma:**

**#pragma TC113\_CPU11 [on | off | restore]**

#### **Assembler control:**

**\$TC113\_CPU11 {on | off}**

#### **Assembler macro:**

The assembler macro `_TC113_CPU11` is defined if you specify the option **-zTC113\_CPU11**.

#### **Protected libraries to link:**

`lib\p\tc113\*.a`

#### **Compiler bypass:**

To bypass this CPU functional problem, the C compiler generates a NOP instruction between a LDA, LDDA, LD16A and the JI instruction. The compiler also generates a NOP before a RET and RET16 instruction if there is no or just one instruction before RET, starting from the function entry point.

#### **Assembler check:**

The assembler gives a warning when an LDA, LDDA, or LD16A instruction is directly followed by a JI instruction. The assembler also gives a warning when there is no or just one instruction (not a NOP instruction) between label and RET or RET16:

```
177 suspicious instruction concerning CPU functional
defect TC113_CPU11
```

You can suppress this warning with the option **-w177**.

### **3.6 TC113\_CPU13**

**Assembler option:**

**-zTC113\_CPU13**

**Assembler macro:**

The assembler macro `_TC113_CPU13` is defined if you specify the option **-zTC113\_CPU13**.

**Protected libraries to link:**

`lib\p\tc113\*.a` (or add `lib\src\cstart.asm` to your project).

**Compiler bypass:**

To bypass this CPU functional problem, an assembler macro is added to the C startup code to enable the 16Kb D-Cache. The DCSIZ bits are set to 16Kb in the SFR register `DMU_CON`.

**Assembler check:**

No assembler check is supported, because run time D-Cache size can not be checked at assembly level. As an alternative, you can use the **TC113\_CPU14** assembler check and workaround. You can also insert a `DSYNC` to workaround this CPU functional problem.

### **3.7 TC113\_CPU14**

#### **Compiler and assembler option:**

**-zTC113\_CPU14**

#### **Pragma:**

**#pragma TC113\_CPU14 [on | off | restore]**

#### **Assembler control:**

**\$TC113\_CPU14 {on | off}**

#### **Assembler macro:**

The assembler macro `_TC113_CPU14` is defined if you specify the option **-zTC113\_CPU14**.

#### **Protected libraries to link:**

`lib\p\tc113\*.a`

#### **Compiler bypass:**

To bypass this CPU functional problem, the C compiler generates a DSYNC instruction directly after a (interrupt) function entry point label. Also an assembler macro is added to the run-time library functions for optionally adding a DSYNC instruction after a function entry point label.

#### **Assembler check:**

The assembler gives a warning when the first label in a code section is not followed by a DSYNC instruction:

```
178 suspicious instruction concerning CPU functional
defect TC113_CPU14
```

You can suppress this warning with the option **-w178**.

### **3.8 TC113\_CPU15**

**Compiler and assembler option:**

**-zTC113\_CPU15**

**Pragma:**

**#pragma TC113\_CPU15 [on | off | restore]**

**Assembler control:**

**\$TC113\_CPU15 {on | off}**

**Assembler macro:**

The assembler macro `_TC113_CPU15` is defined if you specify the option **-zTC113\_CPU15**.

**Protected libraries to link:**

`lib\p\tc113\*.a`

**Compiler bypass:**

To bypass this CPU functional problem, the C compiler avoids generation of the ST.T, SWAP and LDMST instructions. For immediate `_bit` and bit-field operations alternative instructions are used.

**Assembler check:**

The assembler gives a warning for ST.T, SWAP and LDMST instructions:

```
179 suspicious instruction concerning CPU functional
defect TC113_CPU15
```

You can suppress this warning with the option **-w179**.

### **3.9 TC113\_CPU16**

#### **Compiler and assembler option:**

**-zTC113\_CPU16**

#### **Pragma:**

**#pragma TC113\_CPU16 [on | off | restore]**

#### **Assembler control:**

**\$TC113\_CPU16 {on | off}**

#### **Assembler macro:**

The assembler macro `_TC113_CPU16` is defined if you specify the option **-zTC113\_CPU16**.

#### **Protected libraries to link:**

`lib\p\tc113\*.a`

#### **Compiler bypass:**

To bypass this CPU functional problem, the C compiler generates a NOP instruction between an LDA, LDDA, LD16A and the JI or CALLI instruction with the same address register as parameter. The compiler also generates a NOP instruction before a RET and RET16 instruction if there is no or just one instruction before RET, starting from the function entry point.

#### **Assembler check:**

The assembler gives a warning when an LDA, LDDA or LD16A instruction is directly followed by a JI or CALLI instruction with the same address register as parameter. The assembler also gives a warning when there is no or just one instruction (not a NOP instruction) between label and RET or RET16:

```
180 suspicious instruction concerning CPU functional
 defect TC113_CPU16
```

You can suppress this warning with the option **-w180**.

### **3.10 TC113\_DMU1**

**Compiler and assembler option:**

**-zTC113\_DMU1**

**Pragma:**

**#pragma TC113\_DMU1 [on | off | restore]**

**Assembler control:**

**\$TC113\_DMU1 {on | off}**

**Assembler macro:**

The assembler macro `_TC113_DMU1` is defined if you specify the option **-zTC113\_DMU1**.

**Protected libraries to link:**

`lib\p\tc113\*.a`

**Compiler bypass:**

To bypass this CPU functional problem, the C compiler avoids generation of the ST.T, SWAP and LDMST instructions. For direct `_bit` and bit-field operations, alternative instructions are used.

**Assembler check:**

The assembler gives a for SWAP, LDMST and ST.T instructions:

```
181 suspicious instruction concerning CPU functional
defect TC113_DMU1
```

You can suppress this warning with the option **-w181**.



### **3.11 TC113\_LFI2**

#### **Compiler and assembler option:**

**-zTC113\_LFI2**

#### **Pragma:**

**#pragma TC113\_LFI2 [on | off | restore]**

#### **Assembler control:**

**\$TC113\_LFI2 {on | off}**

#### **Assembler macro:**

The assembler macro `_TC113_LFI2` is defined if you specify the option **-zTC113\_LFI2**.

#### **Protected libraries to link:**

`lib\p\tc113\*.a`

#### **Compiler bypass:**

To bypass this CPU functional problem, the C compiler avoids generation of ST.T, SWAP and LDMST instructions. For immediate `_bit` and bit-field operations alternative instructions are used.

#### **Assembler check:**

The assembler gives a warning for SWAP, LDMST and ST.T instructions:

```
182 suspicious instruction concerning CPU functional
defect TC113_LFI2
```

You can suppress this warning with the option **-w182**.

### **3.12 TC113\_LFI3**

**Compiler and assembler option:**

**-zTC113\_LFI3**

**Pragma:**

**#pragma TC113\_LFI3 [on | off | restore]**

**Assembler control:**

**\$TC113\_LFI3 {on | off}**

**Assembler macro:**

The assembler macro `_TC113_LFI3` is defined if you specify the option **-zTC113\_LFI3**.

**Protected libraries to link:**

`lib\p\tc113\*.a`

**Compiler bypass:**

To bypass this CPU functional problem, the compiler avoids generation of the ST.T, SWAP and LDMST instructions. For direct `_bit` and bit-field operations alternative instructions are used.

**Assembler check:**

The assembler gives a warning for SWAP, LDMST and ST.T instructions:

```
183 suspicious instruction concerning CPU functional
defect TC113_LFI3
```

You can suppress this warning with the option **-w183**.

# CPU FUNCTIONAL PROBLEMS

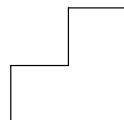
# INDEX

## INDEX

---



**TASKING**



---

# INDEX

---

# Symbols

#define, 4-18  
 #include, 4-28, 4-68  
 #pragma, 4-71  
     *asm*, 3-37  
     *asm\_noflush*, 3-37  
     *clear*, 4-72  
     *endasm*, 3-37  
     *noclear*, 4-72  
 #pragma optimize, 4-34  
 #undef, 4-59  
     \_\_DATE\_\_, 4-59  
     \_\_FILE\_\_, 4-59  
     \_\_LINE\_\_, 4-59  
     \_\_START\_\_, 7-3  
     \_\_STDC\_\_, 4-59  
     \_\_TIME\_\_, 4-59  
     \_a0, storage type, 3-5  
     \_a1, storage type, 3-5  
     \_a8, storage type, 3-5  
     \_a9, storage type, 3-5  
     \_accum, 3-8, 3-13  
     \_at attribute, 3-6  
     \_atbit attribute, 3-7  
     \_bisr\_, 3-28  
     \_bit, 3-8  
     \_circ, 3-46  
     \_close, 6-12  
     \_CTRI, 4-59  
     \_DOUBLE\_FP, 4-60  
     \_enable\_, 3-27  
     \_far, 3-30  
         *storage type*, 3-5  
     \_fp\_get\_exception\_mask, 7-15  
     \_fp\_get\_exception\_status, 7-16  
     \_fp\_install\_trap\_handler, 7-16  
     \_fp\_set\_exception\_mask, 7-15  
     \_fp\_set\_exception\_status, 7-16  
     \_fract, 3-8, 3-13  
     \_inline, 3-36  
     \_interrupt, 3-24  
     \_interrupt\_fast, 3-24

    \_lseek, 6-12  
     \_near, storage type, 3-5  
     \_open, 6-12  
     \_packb, 3-17  
     \_packhw, 3-17  
     \_read, 6-12  
     \_sat, 3-16  
     \_sfract, 3-8, 3-13  
     \_sfrbit16, 3-21  
     \_sfrbit32, \_sfrbit32, 3-21  
     \_SINGLE\_FP, 4-60  
     \_stackparm, 3-29  
     \_syscallfunc, 3-29  
     \_TASKING, 4-59  
     \_tolower, 6-13  
     \_toupper, 6-13  
     \_trap, 3-25  
     \_trap\_fast, 3-25  
     \_unlink, 6-13  
     \_write, 6-13

## A

abort, 6-14  
 abs, 6-14  
 access, 6-14  
 acos, 6-14  
 adding files to a project, 2-20  
 alias, 4-37  
 align, 4-71  
 ansi standard, 2-3, 3-3, 4-59  
 asctime, 6-15  
 asin, 6-15  
 asm, 4-72  
 asm\_noflush, 4-72  
 assembly source file, 2-10  
 assert, 6-15  
 assert.h, 6-3  
     *assert*, 6-15  
 astri, 2-10  
 atan, 6-15  
 atan2, 6-16

atexit, 6-16  
 atof, 6-16  
 atoi, 6-16  
 atol, 6-17

## B

backend  
   *compiler phase*, 2-5  
   *optimization*, 2-5, 2-9  
 bit data type, 3-20  
 bsearch, 6-17  
 buffer, circular, 3-46  
 built-in functions, 3-40

## C

C  
   *inline functions*, 3-36  
   *language extensions*, 3-3  
 C library, 6-4  
   *implementation details*, 6-6  
   *interface description*, 6-12  
   *name syntax*, 6-5  
   *reentrancy*, 6-62  
 C startup code, 7-3  
 cache global variables, 4-52  
 calloc, 6-17  
 ceil, 6-18  
 chdir, 6-18  
 circular buffer, 3-46  
 circular pointer, 3-46  
 clear, 4-72  
 clearerr, 6-18  
 clock, 6-18  
 close, 6-19  
 code generator, 2-6  
 command file, 4-23  
 command line processing, 4-23  
 comments, C++ style, 4-13

common subexpression elimination,  
   2-8  
 compiler, invocation, 4-6  
 compiler limits, 4-78  
 compiler options  
   -?, 4-11  
   -A, 4-12  
   -builtin, 4-15  
   -C, 4-16  
   -c, 4-17  
   -D, 4-18  
   -E, 4-19  
   -e, 4-20  
   -El, 4-19  
   -Em, 4-19  
   -err, 4-21  
   -F, 4-22, 6-6  
   -f, 4-23  
   -Fc, 4-22, 6-6  
   -FPU, 4-25  
   -g, 4-26  
   -gf, 4-26  
   -gl, 4-26  
   -gn, 4-26  
   -H, 4-27  
   -I, 4-28  
   -indirect, 4-29  
   -K, 4-30  
   -misrac, 4-31  
   -N, 4-32  
   -n, 4-33  
   -O, 4-34, 4-36  
   -o, 4-55  
   -Oa / -OA, 4-37  
   -Oc / -OC, 4-38  
   -Oe / -OE, 4-40  
   -Of / -OF, 4-41  
   -Oi / -OI, 4-43  
   -Ol / -OL, 4-44  
   -Oo / -OO, 4-45  
   -Op / -OP, 4-47  
   -Os / -OS, 4-48

- Ot / -OT, 4-49
- Ou / -OU, 4-50
- Ov / -OV, 4-51
- Ow / -OW, 4-52
- Oy / -OY, 4-53
- Oz / -OZ, 4-54
- R, 4-56
- s, 4-57
- TC2, 4-58
- U, 4-59
- u, 4-61
- V, 4-62
- w, 4-64
- WAE, 4-63
- wstrict, 4-64
- Z, 4-65
- z, 4-66
- detailed description*, 4-10
- overview*, 4-6
- overview in functional order*, 4-8
- priority*, 4-6
- compiler phases, 2-5
  - backend*, 2-5
  - code generator phase*, 2-6
  - optimization phase*, 2-5
  - peephole optimizer phase*, 2-6
  - pipeline scheduler*, 2-6
- frontend*, 2-5
  - optimization phase*, 2-5
  - parser phase*, 2-5
  - preprocessor phase*, 2-5
  - scanner phase*, 2-5
- compiler structure, 2-10
- compound assignment, 4-40
- conditional jump reversal, 2-7, 4-41
- constant folding, 2-6
- constant propagation, 2-7, 4-45, 4-47
- control flow optimization, 2-7, 4-41
- control program, 4-3
  - options overview*, 4-4
- conversions
  - ANSI C, 3-9
  - fractional*, 3-13

- copy propagation, 2-7, 4-47
- copysign, 6-19
- cos, 6-19
- cosh, 6-19
- creating a makefile, 2-21
- cross-assembler, 2-10
- CSE, 2-8, 4-38
- cstart.asm, 7-3
- ctime, 6-20
- CTRINC, 4-28, 4-68
- ctype.h, 6-3
  - \_tolower*, 6-13
  - \_toupper*, 6-13
  - isalnum*, 6-29
  - isalpha*, 6-29
  - isascii*, 6-29
  - iscntrl*, 6-29
  - isdigit*, 6-30
  - isgraph*, 6-30
  - islower*, 6-30
  - isprint*, 6-31
  - ispunct*, 6-31
  - isspace*, 6-31
  - isupper*, 6-31
  - isxdigit*, 6-32
  - toascii*, 6-58
  - tolower*, 6-58
  - toupper*, 6-58

## D

- data types, 3-8-3-12
  - \_accum*, 3-8, 3-13
  - \_bit*, 3-8
  - \_fract*, 3-8, 3-13
  - \_packb*, 3-17
  - \_packbw*, 3-17
  - \_sfract*, 3-8, 3-13
  - double*, 3-8
  - enum*, 3-8
  - float*, 3-8



*fractional*, 3-13-3-15  
*packed*, 3-17-3-20  
*pointer*, 3-8  
*signed char*, 3-8, 3-9  
*signed int*, 3-8  
*signed long*, 3-8  
*signed short*, 3-8  
*unsigned char*, 3-8, 3-9  
*unsigned int*, 3-8  
*unsigned long*, 3-8  
*unsigned short*, 3-8  
 dead assignment elimination, 2-8  
 dead code elimination, 2-8  
 dead storage elimination, 2-8  
 debug information, 4-26  
 debugger, starting, 2-19  
 derivatives, 2-4  
 detailed option description, compiler,  
     4-10-4-67  
 development flow, 2-10  
 difftime, 6-20  
 directory separator, 4-69  
 div, 6-20  
 double, 3-8

## E

EDE, 2-14  
     *build an application*, 2-18  
     *load files*, 2-16  
     *open a project*, 2-16  
     *select a toolchain*, 2-15  
     *start a new project*, 2-20  
     *starting*, 2-14  
 embedded development environment.  
     *See* EDE  
 enabling MISRA C, 3-44  
 endasm, 4-72  
 enum, 3-8  
 environment variable  
     *CTRIINC*, 4-28, 4-68  
     *LM\_LICENSE\_FILE*, 1-17, A-6

*overview of*, 2-13  
     *PATH*, 1-4, 1-7, 1-10  
     *TMPDIR*, 1-4, 1-7, 1-10  
     *used by tool chain*, 2-13  
 errno declaration, 6-70  
 errno.h, 6-3, 6-70  
 error level, 5-4  
 Error Messages, 3-45  
 errors, 5-5  
     *backend*, 5-32  
     *FLEXlm license*, A-33  
     *frontend*, 5-5  
 example  
     *starting EDE*, 2-14  
     *using EDE*, 2-14  
     *using the control program*, 2-21  
     *using the makefile*, 2-23  
 exceptions, floating point, 7-13  
 execution time, 3-23  
 exit, 6-20  
 exit status, 5-4  
 exp, 6-21  
 expression propagation, 4-40  
 expression rearrangement, 2-6  
 expression simplification, 2-7  
 extensions to C, 3-3

## F

fabs, 6-21  
 FAQ, FLEXlm, A-37  
 fast code, 4-48  
 fast loops, 4-44  
 fclose, 6-21  
 fcntl.h, 6-3  
     *open*, 6-38  
 feof, 6-21  
 ferror, 6-21  
 fflush, 6-22  
 fgetc, 6-22  
 fgetpos, 6-22  
 fgets, 6-22

file system simulation, 6-3  
 Flexible License Manager, A-1  
 FLEXlm, A-1  
     *daemon log file*, A-25  
     *daemon options file*, A-7  
     *FAQ*, A-37  
     *frequently asked questions*, A-37  
     *license administration tools*, A-8  
     *for Windows*, A-22  
     *license errors*, A-33  
 float, 3-8  
 float.h, 6-3  
     *copysign*, 6-19  
     *isfinite*, 6-30  
     *isinf*, 6-30  
     *isnan*, 6-31  
     *scalb*, 6-44  
 floating license, 1-11  
 floating point, 7-10  
     *single precision*, 4-25, 6-6  
     *special values*, 7-13  
     *trap handler*, 7-14  
     *trap handling api*, 7-15  
     *trapping*, 7-13  
 floating point constants, 4-22  
 floor, 6-23  
 fmod, 6-23  
 fopen, 6-23  
 fprintf, 6-24  
 fputc, 6-24  
 fputs, 6-24  
 fractional data types, 3-13  
     *intrinsic functions*, 3-15  
     *operations on*, 3-14  
     *promotion rules*, 3-14  
 fread, 6-25  
 free, 6-25  
 freopen, 6-25  
 frexp, 6-26  
 frontend  
     *compiler phase*, 2-5  
     *optimization*, 2-5, 2-6  
 fscanf, 6-26

fseek, 6-26  
 fsetpos, 6-27  
 fss.h, 6-3  
 ftell, 6-27  
 function qualifier  
     *\_bstr\_*, 3-28  
     *\_enable\_*, 3-27  
     *\_far*, 3-30  
     *\_interrupt*, 3-24  
     *\_interrupt\_fast*, 3-24  
     *\_stackparm*, 3-29  
     *\_syscallfunc*, 3-29  
     *\_trap*, 3-25  
     *\_trap\_fast*, 3-25  
 function return types, 7-7  
 functional problems, C-3  
 functions  
     *built-in*, 3-40  
     *intrinsic*, 3-40  
 fwrite, 6-27

## G

getc, 6-27  
 getchar, 6-28  
 getcwd, 6-28  
 getenv, 6-28  
 gets, 6-28  
 global variables, 4-52  
 gmtime, 6-29

## H

header files, 6-3  
 heap, 7-3, 7-10  
     *begin of*, 7-10  
     *end of*, 7-10  
 heap size, 7-10  
 hostid, determining, 1-19  
 hostname, determining, 1-19

identifier, 4-13

IEEE 32-bit single precision format,  
3-8

IEEE 64-bit double precision format,  
3-8

IEEE-695, 2-12

in-line functions, 2-8

include files, 4-68

*default directory*, 4-69

indirect function calling, 4-29

inline assembly, 3-37

installation

*licensing*, 1-11

*Linux*, 1-5

*RPM*, 1-5

*tar.gz*, 1-6

*UNIX*, 1-8

*Windows*, 1-3

*Windows 95*, 1-3

*Windows NT*, 1-3

instruction scheduling, 4-54

integral promotion, 3-9

Intel hex format, 2-12

intenum, 4-72

function, inline C, 3-36

interrupts, 3-24

intrinsic functions, 3-40

*\_abs*, 3-41

*\_absb*, 3-19

*\_absb*, 3-19

*\_abss*, 3-41

*\_abssb*, 3-19

*\_bshr*, 3-41

*\_clo*, 3-41

*\_cls*, 3-41

*\_clz*, 3-41

*\_debug*, 3-41

*\_disable*, 3-40

*\_dsync*, 3-41

*\_enable*, 3-40

*\_extr*, 3-42

*\_extractbyte1*, 3-18

*\_extractbyte2*, 3-18

*\_extractbyte3*, 3-18

*\_extractbyte4*, 3-18

*\_extractbw1*, 3-18

*\_extractbw2*, 3-18

*\_extru*, 3-42

*\_getbit*, 3-43

*\_getbyte1*, 3-18

*\_getbyte2*, 3-18

*\_getbyte3*, 3-18

*\_getbyte4*, 3-18

*\_getfract*, 3-15

*\_gethw1*, 3-18

*\_gethw2*, 3-18

*\_imaskldmst*, 3-42

*\_initpackb*, 3-18

*\_initpackbl*, 3-17

*\_initpackbw*, 3-18

*\_initpackbwl*, 3-17

*\_ins*, 3-42

*\_insert*, 3-42

*\_insertbyte1*, 3-18

*\_insertbyte2*, 3-18

*\_insertbyte3*, 3-18

*\_insertbyte4*, 3-18

*\_insertbw1*, 3-18

*\_insertbw2*, 3-18

*\_insn*, 3-42

*\_isync*, 3-41

*\_max*, 3-41

*\_maxs*, 3-41

*\_maxu*, 3-41

*\_mfcrr*, 3-41

*\_min*, 3-41

*\_minb*, 3-19

*\_minbu*, 3-19

*\_minb*, 3-19

*\_mins*, 3-41

*\_minu*, 3-41

*\_mtrr*, 3-41

- \_mulfractlong*, 3-15
- \_mulsc*, 3-42
- \_nop*, 3-41
- \_parity*, 3-41
- \_putbit*, 3-42
- \_restore*, 3-40
- \_round16*, 3-15
- \_rslcx*, 3-41
- \_sath*, 3-41
- \_satbu*, 3-41
- \_sath*, 3-41
- \_satbu*, 3-41
- \_setbyte1*, 3-18
- \_setbyte2*, 3-18
- \_setbyte3*, 3-18
- \_setbyte4*, 3-18
- \_sethw1*, 3-18
- \_sethw2*, 3-18
- \_shaaccum*, 3-15
- \_shafracts*, 3-15
- \_sbafracts*, 3-15
- \_svlcx*, 3-41
- \_swapmsk*, 3-19
- \_syscall*, 3-41
- \_transpose\_byte*, 3-18
- \_transpose\_hword*, 3-18
- display prototypes*, 4-15
- fractional*, 3-15
- packed*, 3-17
- introduction, 2-3
- invariant code, 2-8, 4-43
- invocation
  - compiler*, 4-6
  - control program*, 4-3
- iob structures, 6-69
- isalnum, 6-29
- isalpha, 6-29
- isascii, 6-29
- iscntrl, 6-29
- isdigit, 6-30
- isfinite, 6-30
- isgraph, 6-30
- isinf, 6-30

- islower, 6-30
- isnan, 6-31
- isprint, 6-31
- ispunct, 6-31
- isspace, 6-31
- isupper, 6-31
- isxdigit, 6-32

## J

- jump chain, 3-47
- jump chaining, 2-7, 4-41
- jump table, 3-47

## K

- keyword, *\_inline*, 3-36

## L

- labs, 6-32
- language extensions, 4-12
- lctri, 2-10
- ldexp, 6-32
- ldiv, 6-32
- leaf function handling, 2-9
- libraries
  - C*, 6-4
  - C (single precision floating point)*, 6-6
  - floating point*, 6-5
  - name syntax*, 6-5
  - run-time*, 6-72
- license
  - floating*, 1-11
  - node-locked*, 1-11
  - obtaining*, 1-11
- license file
  - default location*, A-6

*location*, 1-17  
 licensing, 1-11  
 limits, compiler, 4-78  
 limits.h, 6-3  
 linker, 2-10  
 listinc, 4-73  
 lktri, 2-10  
 LM\_LICENSE\_FILE, 1-17, A-6  
 lmcksum, A-10  
 lmdiag, A-11  
 lmdown, A-12  
 lmgrd, A-13  
 lmhostid, A-15  
 lmremove, A-16  
 lmreread, A-17  
 lmstat, A-18  
 lmswitchr, A-20  
 lmver, A-21  
 locale.h, 6-3  
     *localeconv*, 6-33  
     *setlocale*, 6-47  
 localeconv, 6-33  
 localtime, 6-33  
 locator, 2-10  
 log, 6-33  
 log10, 6-33  
 logical expression optimization, 2-7  
 longjmp, 6-34  
 lookup table, 3-47  
 loop rotation, 2-7, 4-44  
 loop unrolling, 2-8, 4-50  
 loop variable detection, 4-38  
 lseek, 6-34

## M

makefile  
     *automatic creation of*, 2-21  
     *updating*, 2-21  
 malloc, 6-34  
 malloc.h, 6-3

math.h, 6-3  
     *acos*, 6-14  
     *asin*, 6-15  
     *atan*, 6-15  
     *atan2*, 6-16  
     *ceil*, 6-18  
     *cos*, 6-19  
     *cosh*, 6-19  
     *exp*, 6-21  
     *fabs*, 6-21  
     *floor*, 6-23  
     *fmod*, 6-23  
     *frexp*, 6-26  
     *ldexp*, 6-32  
     *log*, 6-33  
     *log10*, 6-33  
     *modf*, 6-37  
     *pow*, 6-38  
     *sin*, 6-48  
     *sinh*, 6-48  
     *sqrt*, 6-49  
     *tan*, 6-57  
     *tanh*, 6-57  
 mblen, 6-35  
 mbstowcs, 6-35  
 mbtowc, 6-35  
 memchr, 6-36  
 memcmp, 6-36  
 memcpy, 6-36  
 memmove, 6-36  
 memory access, 3-4  
 memset, 6-37  
 MISRA C, 3-44, B-1, C-1  
 mktime, 6-37  
 modf, 6-37  
 Motorola S-record, 2-12  
 multi-line macros, 4-19

# N

name syntax, C library, 6-5  
 noclear, 4-72  
 node-locked license, 1-11  
 nolistinc, 4-73  
 nosource, 4-76

# O

offsetof, 6-37  
 open, 6-38  
 optimization, 4-34, 4-36  
   *backend*, 2-5, 2-9  
   *frontend*, 2-5, 2-6  
 optimization (backend)  
   *leaf function handling*, 2-9  
   *peephole optimizations*, 2-9  
   *tail recursion elimination*, 2-9  
 optimization (frontend)  
   *common subexpression elimination*,  
     2-8  
   *conditional jump reversal*, 2-7  
   *constant folding*, 2-6  
   *constant/copy propagation*, 2-7  
   *control flow optimization*, 2-7  
   *dead assignment elimination*, 2-8  
   *dead code elimination*, 2-8  
   *dead storage elimination*, 2-8  
   *expression rearrangement*, 2-6  
   *expression simplification*, 2-7  
   *in-line functions*, 2-8  
   *invariant code motion*, 2-8  
   *jump chaining*, 2-7  
   *logical expression optimization*, 2-7  
   *loop rotation*, 2-7  
   *loop unrolling*, 2-8  
   *remove useless jumps*, 2-7  
   *sharing of string literals and floating*  
     *point constants*, 2-8  
   *subscript strength reduction*, 2-8

optimize, 4-73  
 optimize restore, 4-73  
 options, control program, 4-4  
 output file, 4-55

# P

pack 0, 4-74  
 pack 2, 4-73  
 packed data types, 3-17  
   *intrinsic functions*, 3-17  
 parameter passing, 3-23  
 parser, 2-5  
 PATH, 1-4, 1-7, 1-10  
 peephole optimization, 2-9, 4-53  
 peephole optimizer, 2-6  
 perror, 6-38  
 pipeline scheduler, 2-6  
 pipeline scheduling, 4-54  
 pointer, 3-8  
 pow, 6-38  
 power-on vector, 7-3  
 pragma  
   *align*, 4-71  
   *asm*, 4-72  
   *asm\_noflush*, 4-72  
   *clear*, 4-72  
   *endasm*, 4-72  
   *intenum*, 4-72  
   *listinc*, 4-73  
   *noclear*, 4-72  
   *nolistinc*, 4-73  
   *nosource*, 4-76  
   *on command line*, 4-66  
   *optimize*, 4-73  
   *optimize restore*, 4-73  
   *pack 0*, 4-74  
   *pack 2*, 4-73  
   *section*, 4-74  
   *source*, 4-76  
   *stack*, 4-76

*switch**auto*, 4-76*jumpstab*, 4-76*linear*, 4-76*lookup*, 4-76*restore*, 4-76*TC112\_DETECTS*, 4-77*TC113\_DETECTS*, 4-77

pragma optimize, 4-34

pragmas, 4-71

predefined symbols, 4-59

*\_CTRL*, 4-59*\_DOUBLE\_FP*, 4-60*\_SINGLE\_FP*, 4-60*\_TASKING*, 4-59

printf, 6-39

product definition, 2-4

project files, adding files, 2-20

putc, 6-41

putchar, 6-41

puts, 6-41

**Q**

qsort, 6-42

**R**

raise, 6-42

RAM, 3-4

rand, 6-42

read, 6-42

realloc, 6-43

reentrancy, 6-62

register usage, 7-7

remove useless jumps, 2-7

rename, 6-43

restrict, 3-33

return values, 5-4

rewind, 6-44

run-time library, 6-72

**S**

sample session, 2-14

scalb, 6-44

scanf, 6-44

scanner, 2-5

section, 3-6

*rename*, 4-74

section name, 4-56

setbuf, 6-46

setjmp, 6-46

setjmp.h, 6-3

*longjmp*, 6-34*setjmp*, 6-46

setlocale, 6-47

setting the environment, 1-4, 1-7, 1-10

setvbuf, 6-47

sharing of string literals and floating

point constants, 2-8

SIGABRT, 6-48

SIGFPE, 6-48

SIGFPE signal handler, 7-14

SIGILL, 6-48

SIGINT, 6-48

signal, 6-48

signal.h, 6-3

*raise*, 6-42*signal*, 6-48

signals, 6-48

signed

*char*, 3-8, 3-9*int*, 3-8*long*, 3-8*short*, 3-8

signed characters, 3-9

SIGSEGV, 6-48

SIGTERM, 6-48

sin, 6-48

sinh, 6-48

software pipelining, 4-45

source, 4-76

sprintf, 6-49

sqrt, 6-49

rand, 6-49  
sscanf, 6-49  
stack, 4-76, 7-9  
    *begin of*, 7-9  
stack size, 7-9  
start.obj, 7-3  
startup code, 7-3  
stat, 6-50  
stdarg.h, 6-3  
    *va\_arg*, 6-59  
    *va\_end*, 6-59  
    *va\_start*, 6-60  
stddef.h, 6-4  
    *offsetof*, 6-37  
stdio.h, 6-4  
    *\_close*, 6-12  
    *\_lseek*, 6-12  
    *\_open*, 6-12  
    *\_read*, 6-12  
    *\_unlink*, 6-13  
    *\_write*, 6-13  
    *clearerr*, 6-18  
    *fclose*, 6-21  
    *feof*, 6-21  
    *ferror*, 6-21  
    *fflush*, 6-22  
    *fgetc*, 6-22  
    *fgetpos*, 6-22  
    *fgets*, 6-22  
    *fopen*, 6-23  
    *fprintf*, 6-24  
    *fputc*, 6-24  
    *fputs*, 6-24  
    *fread*, 6-25  
    *freopen*, 6-25  
    *fscanf*, 6-26  
    *fseek*, 6-26  
    *fsetpos*, 6-27  
    *ftell*, 6-27  
    *fwrite*, 6-27  
    *getc*, 6-27  
    *getchar*, 6-28  
    *gets*, 6-28  
    *perror*, 6-38  
    *printf*, 6-39  
    *putc*, 6-41  
    *putchar*, 6-41  
    *puts*, 6-41  
    *remove*, 6-43  
    *rename*, 6-43  
    *rewind*, 6-44  
    *scanf*, 6-44  
    *setbuf*, 6-46  
    *setvbuf*, 6-47  
    *sprintf*, 6-49  
    *sscanf*, 6-49  
    *tmpfile*, 6-57  
    *tmpnam*, 6-58  
    *ungetc*, 6-59  
    *vfprintf*, 6-60  
    *vprintf*, 6-60  
    *vsprintf*, 6-60  
stdlib.h, 6-4  
    *abort*, 6-14  
    *abs*, 6-14  
    *atexit*, 6-16  
    *atof*, 6-16  
    *atoi*, 6-16  
    *atol*, 6-17  
    *bsearch*, 6-17  
    *calloc*, 6-17  
    *div*, 6-20  
    *exit*, 6-20  
    *free*, 6-25  
    *getenv*, 6-28  
    *labs*, 6-32  
    *ldiv*, 6-32  
    *malloc*, 6-34  
    *mblen*, 6-35  
    *mbstowcs*, 6-35  
    *mbtowc*, 6-35  
    *qsort*, 6-42  
    *rand*, 6-42  
    *realloc*, 6-43  
    *srand*, 6-49  
    *strtod*, 6-55



- strtol*, 6-55
- strtoul*, 6-56
- system*, 6-56
- wcstombs*, 6-61
- wctomb*, 6-61
- storage type, 3-4
  - \_a0*, 3-5
  - \_a1*, 3-5
  - \_a8*, 3-5
  - \_a9*, 3-5
  - \_far*, 3-5
  - \_near*, 3-5
- strcat*, 6-50
- strchr*, 6-50
- strcmp*, 6-50
- strcoll*, 6-51
- strcpy*, 6-51
- strcspn*, 6-51
- strerror*, 6-51
- strftime*, 6-52
- string*, 3-34
- string.h*, 6-4
  - memchr*, 6-36
  - memcmp*, 6-36
  - memcpy*, 6-36
  - memmove*, 6-36
  - memset*, 6-37
  - strcat*, 6-50
  - strchr*, 6-50
  - strcmp*, 6-50
  - strcoll*, 6-51
  - strcpy*, 6-51
  - strcspn*, 6-51
  - strerror*, 6-51
  - strlen*, 6-53
  - strncat*, 6-53
  - strncmp*, 6-53
  - strncpy*, 6-53
  - strpbrk*, 6-54
  - strrchr*, 6-54
  - strspn*, 6-54
  - strstr*, 6-54
  - strtok*, 6-55
  - strxfrm*, 6-56
  - strlen*, 6-53
  - strncat*, 6-53
  - strncmp*, 6-53
  - strncpy*, 6-53
  - strpbrk*, 6-54
  - strrchr*, 6-54
  - strspn*, 6-54
  - strstr*, 6-54
  - strtod*, 6-55
  - strtok*, 6-55
  - strtol*, 6-55
  - strtoul*, 6-56
  - structure tag, 3-45
  - strxfrm*, 6-56
  - subscript strength reduction, 2-8, 4-51
  - switch
    - auto*, 3-47, 4-76
    - jumpstab*, 3-47, 4-76
    - linear*, 3-47, 4-76
    - lookup*, 3-47, 4-76
    - restore*, 3-47, 4-76
  - switch statement, 3-47-3-48
  - symbols, predefined, 4-59
  - system, 6-56

## T

- tail call conversion, 4-49
- tail recursion elimination, 2-9
- tan*, 6-57
- tanh*, 6-57
- target processors, 2-4
- TC112\_DETECTS, 4-77
- TC113\_DETECTS, 4-77
- time*, 6-57
- time.h*, 6-4
  - asctime*, 6-15
  - clock*, 6-18
  - ctime*, 6-20
  - difftime*, 6-20

- gmtime*, 6-29
- localtime*, 6-33
- mktime*, 6-37
- strftime*, 6-52
- time*, 6-57
- TMPDIR, 1-4, 1-7, 1-10
- tmpfile, 6-57
- tmpnam, 6-58
- toascii, 6-58
- tolower, 6-58
- toupper, 6-58
- transferring parameters between
  - functions, 3-23
- trap, 7-18
- trap handler, 7-14
- trap handling api, 7-15
- TriCore2 instructions, 4-58
- type checking, 4-30
  - disable*, 4-26
- type conversion, fractional, 3-13
- type qualifier
  - \_restrict*, 3-33
  - \_sat*, 3-16
  - \_sfrbit16*, 3-21
  - \_sfrbit32*, 3-21
  - restrict*, 3-33
  - volatile*, 3-32
- typedef, 3-45

## U

- ungetc, 6-59
- unistd.h, 6-4
  - access*, 6-14
  - chdir*, 6-18
  - close*, 6-19

- getcwd*, 6-28
- lseek*, 6-34
- read*, 6-42
- stat*, 6-50
- unlink*, 6-59
- write*, 6-61
- unlink, 6-59
- unsigned
  - char*, 3-8, 3-9
  - int*, 3-8
  - long*, 3-8
  - short*, 3-8
- updating makefile, 2-21

## V

- va\_arg*, 6-59
- va\_end*, 6-59
- va\_start*, 6-60
- variable argument list, 3-35
- version information, 4-62
- vfprintf*, 6-60
- volatile, 3-32
- vprintf*, 6-60
- vsprintf*, 6-60

## W

- warnings, 5-5
- warnings (suppress), 4-64
- warnings as errors, 4-63
- wcstombs, 6-61
- wctomb, 6-61
- write, 6-61
- writeback caching, 4-52

