Sourcery™ CodeBench Lite

MIPS ELF

Sourcery CodeBench Lite 2016.05-7

Getting Started

Sourcery™ CodeBench Lite: MIPS ELF: Sourcery CodeBench Lite 2016.05-7: Getting Started

Mentor Graphics®, Inc.

Mentor Graphics Corporation
8005 S.W. Boeckman Road,
Wilsonville, Oregon 97070-7777

Contacting Mentor Graphics Corporation
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
SupportNet: supportnet.mentor.com

Send Feedback on Documentation: supportnet.mentor.com/doc_feedback_form

Abstract

This guide explains how to install and use Sourcery CodeBench Lite, CodeSourcery's customized and validated version of the GNU Toolchain. Sourcery CodeBench Lite includes everything you need for application development, including C and C++ compilers, assemblers, linkers, libraries, and debugging tools.

Table of Contents

Preface iv
1. Intended Audience v
2. Organization v
3. Typographical Conventions vi
1. Installation and Configuration 1
1.1. Terminology 2
1.2. System Requirements 2
1.3. Registering with the Sourcery CodeBench Portal 3
1.4. Downloading an Installer 3
1.5. Installing Sourcery CodeBench Lite 4
1.6. Installing Sourcery CodeBench Lite Updates 7
1.7. Setting up the Environment 7
1.8. Uninstalling Sourcery CodeBench Lite 9
2. Sourcery CodeBench Lite for MIPS ELF 10
2.1. Included Components and Features 11
2.2. Target Library Configurations 11
2.3. CS3 Support 13
2.4. Using Sourcery CodeBench with MIPS Boards 13
2.5. Using Sourcery CodeBench with YAMON 14
2.6. Profiling Support 14
2.7. Using Flash Memory 14
3. Using Sourcery CodeBench from the Command Line 16
3.1. Building an Application 17
3.2. Running Applications on the Target System 17
3.3. Running Applications from GDB 17
4. CS3: The CodeSourcery Common Startup Code Sequence 20
4.1. Linker Scripts 21
4.2. Program Startup and Termination 23
4.3. Memory Layout 26
4.4. Interrupt Vectors and Handlers 28
4.5. Supported Boards for MIPS ELF 28
5. Sourcery CodeBench Debug Sprite 31
5.1. Probing for Debug Devices 32
5.2. Debug Sprite Example 32
5.3. Invoking Sourcery CodeBench Debug Sprite 33
5.4. Sourcery CodeBench Debug Sprite Options 34
5.5. MDI Devices 34
5.6. Debugging a Remote Board 36
5.7. Supported Board Files 37
5.8. Board File Syntax 37
6. Next Steps with Sourcery CodeBench 41
6.1. Sourcery CodeBench Knowledge Base 42
6.2. Manuals for GNU Toolchain Components 42
A. Sourcery CodeBench Lite Release Notes 43
A.1. Changes in Sourcery CodeBench Lite for MIPS ELF 44
B. Sourcery CodeBench Lite Licenses 48
B.1. Sourcery CodeBench Lite License Agreement 49
B.2. Licenses and Third-Party Information for Sourcery CodeBench Lite Components 59

Preface

This preface introduces the Sourcery CodeBench Lite Getting Started guide. It explains the structure of this guide and describes the documentation conventions used.

1. Intended Audience

This guide is written for people who will install and/or use Sourcery CodeBench Lite. Parts of this document assume that you have some familiarity with using the command-line interface.

2. Organization

This document is organized into the following chapters and appendices:

Chapter 1, "Installation and Configuration"	This chapter describes how to download, install and configure Sourcery CodeBench Lite. This section documents host system requirements and explains how to set up your environment so that you can build and debug applications.
Chapter 2, "Sourcery CodeBench Lite for MIPS ELF"	This chapter contains information about using Sourcery CodeBench Lite that is specific to MIPS ELF targets. You should read this chapter to learn how to best use Sourcery CodeBench Lite on your target system.
Chapter 3, "Using Sourcery CodeBench from the Command Line"	This chapter explains how to build applications with Sourcery CodeBench Lite using the command line. In the process of reading this chapter, you will build a simple application that you can use as a model for your own programs.
Chapter 4, "CS3: The CodeSourcery Common Startup Code Sequence"	CS3 is CodeSourcery's low-level board support library. This chapter documents the boards supported by Sourcery CodeBench Lite and the compiler and linker options you need to use with them. It also explains how you can use and modify CS3-provided definitions for memory maps, system startup code and interrupt vectors in your own code.
Chapter 5, "Sourcery CodeBench Debug Sprite"	This chapter describes the use of the Sourcery CodeBench Debug Sprite for remote debugging. The Sprite allows you to debug programs running on a bare board without an operating system. This chapter includes information about the debugging devices and boards supported by the Sprite for MIPS ELF.
Chapter 6, "Next Steps with Sourcery CodeBench"	This chapter describes where you can find additional documentation and information about using Sourcery CodeBench Lite and its components.
Appendix A, "Sourcery CodeBench Lite Release Notes"	This appendix contains information about changes in this release of Sourcery CodeBench Lite for MIPS ELF. You should read through these notes to learn about new features and bug fixes.
Appendix B, "Sourcery CodeBench Lite Licenses"	This appendix provides information about the software licenses that apply to Sourcery CodeBench Lite. Read this appendix to understand your legal rights and obligations as a user of Sourcery CodeBench Lite.

3. Typographical Conventions

The following typographical conventions are used in this guide:

`> command arg ...`	A command, typed by the user, and its output. The “>” character is the command prompt.
`command`	The name of a program, when used in a sentence, rather than in literal input or output.
`literal`	Text provided to or received from a computer program.
`placeholder`	Text that should be replaced with an appropriate value when typing a command.
`\`	At the end of a line in command or program examples, indicates that a long line of literal input or output continues onto the next line in the document.

Chapter 1
Installation and Configuration

This chapter explains how to install Sourcery CodeBench Lite. You will learn how to:

1. Verify that you can install Sourcery CodeBench Lite on your system.
2. Download the appropriate Sourcery CodeBench Lite installer.
3. Configure your environment so that you can use Sourcery CodeBench Lite.
4. Install Sourcery CodeBench Lite.

1.1. Terminology

Throughout this document, the term *host system* refers to the system on which you run Sourcery CodeBench while the term *target system* refers to the system on which the code produced by Sourcery CodeBench runs. The target system for this version of Sourcery CodeBench is `mips-sde-elf`.

If you are developing a workstation or server application to run on the same system that you are using to run Sourcery CodeBench, then the host and target systems are the same. On the other hand, if you are developing an application for an embedded system, then the host and target systems are probably different.

1.2. System Requirements

1.2.1. Host Operating System Requirements

This version of Sourcery CodeBench supports the following host operating systems and architectures:

- Microsoft Windows:
  - Windows 7: 32-bit and 64-bit
  - Windows 8: 32-bit and 64-bit
- Linux®:
  - Ubuntu Desktop 12.04: 64-bit only
  - Ubuntu Desktop 14.04: 32-bit and 64-bit
  - Ubuntu Desktop 15.04: 64-bit only
  - Redhat Enterprise Linux 6.5 and 6.6: 64-bit only
  - Centos 7 and 7.1: 64-bit only
  - SUSE Enterprise Linux Desktop 11 sp3 and 12: 64-bit only

Sourcery CodeBench is built as a 32-bit application. Therefore, even when running on a 64-bit host system, Sourcery CodeBench requires 32-bit host libraries. If these libraries are not already installed on your system, you must install them before installing and using Sourcery CodeBench Lite. Consult your operating system documentation for more information about obtaining these libraries.

1.2.2. Host Hardware Requirements

The amount of disk space required for a complete Sourcery CodeBench Lite installation directory depends on the host operating system and the number of target libraries included. When you start the graphical installer, it checks whether there is sufficient disk space before beginning to install. Note that the graphical installer also requires additional temporary disk space during the installation process. On Microsoft Windows hosts, the installer uses the location specified by the `TEMP` environment variable for these temporary files. If there is not enough free space on that volume, the installer prompts for an alternate location. On Linux hosts, the installer puts temporary files in the directory specified by the `TMP` environment variable, or `/tmp` if that is not set.

1.2.3. Target System Requirements

See Chapter 2, “Sourcery CodeBench Lite for MIPS ELF” for requirements that apply to the target system.

1.3. Registering with the Sourcery CodeBench Portal

If you do not already have a Sourcery CodeBench Portal account, you must register for one now. You must have an active Sourcery CodeBench Lite subscription to download an installer. Evaluation subscriptions are available at no charge and also give you access to support from CodeSourcery.

If you purchased Sourcery CodeBench Lite directly from Mentor Graphics, you already have an account, and you may skip ahead to the next section. However, if you received Sourcery CodeBench Lite with a hardware development kit or from a distributor, you probably do not have an account.

To register for an account, visit the Sourcery CodeBench Portal[1]. Click on the link to register for an evaluation subscription. Follow the instructions on the web site to create your account. Then, once your account is active, click the button to request an evaluation subscription.

You should request an evaluation version of Sourcery CodeBench that matches the version you received with your development kit. Select the host system where you will install Sourcery CodeBench, and MIPS ELF as the target system where you will run applications. Then click the `Request Evaluation` button.

If there are newer versions of Sourcery CodeBench Lite than the one provided with your development kit, they will be visible through the Sourcery CodeBench Portal once your evaluation subscription is active. CodeSourcery recommends that you first work with the version of Sourcery CodeBench that came with your development kit, since CodeSourcery and the manufacturer have tested that particular combination of hardware and software. However, you may also wish to experiment with newer versions.

1.4. Downloading an Installer

If you have received Sourcery CodeBench Lite on a CD, or other physical media, then you do not need to download an installer. You may skip ahead to Section 1.5, “Installing Sourcery CodeBench Lite”.

You can download Sourcery CodeBench Lite from the Sourcery CodeBench web site[2]. This free version of Sourcery CodeBench, which is made available to the general public, does not include all the functionality of CodeSourcery's product releases. If you prefer, you may instead purchase or register for an evaluation of CodeSourcery's product toolchains at the Sourcery CodeBench Portal[3].

Once you have navigated to the appropriate web site, download the installer that corresponds to your host operating system. For Microsoft Windows systems, the Sourcery CodeBench installer is provided as a compressed archive with the `.zip` extension. For GNU/Linux systems Sourcery CodeBench Lite is provided as an executable installer package with the `.bin` extension.

[1] https://sourcery.mentor.com/GNUToolchain/
[2] http://go.mentor.com/codebench/
[3] https://sourcery.mentor.com/GNUToolchain/

On Microsoft Windows systems, save the installer to the desktop. On GNU/Linux systems, save the download package in your home directory.

1.5. Installing Sourcery CodeBench Lite

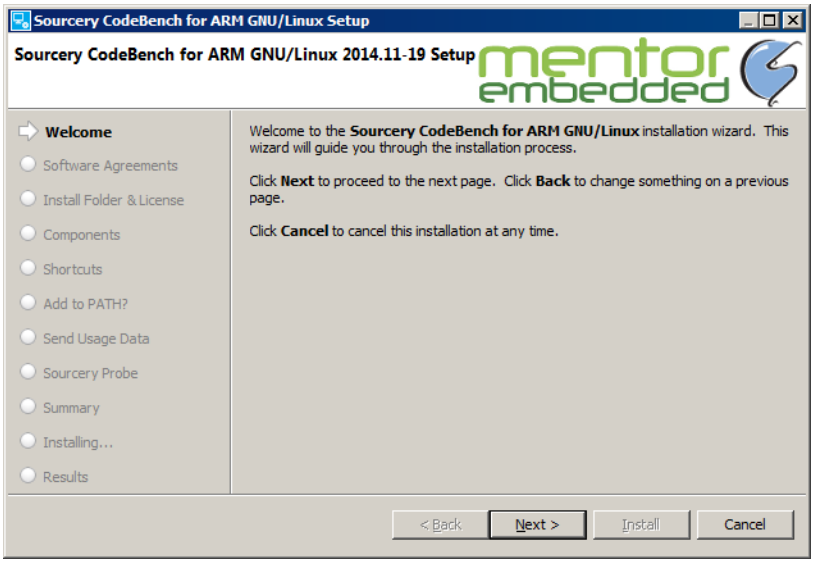
The method used to install Sourcery CodeBench Lite depends on your host system and the kind of installation package you have downloaded.

1.5.1. Using the Sourcery CodeBench Lite Installer on Microsoft Windows

If you have received Sourcery CodeBench Lite on CD, insert the CD in your computer. and double click on the CD. If you downloaded Sourcery CodeBench Lite, double-click on the installer. Extract the `.zip`. Browse to the folder where you extracted the installer and run the installer executable.

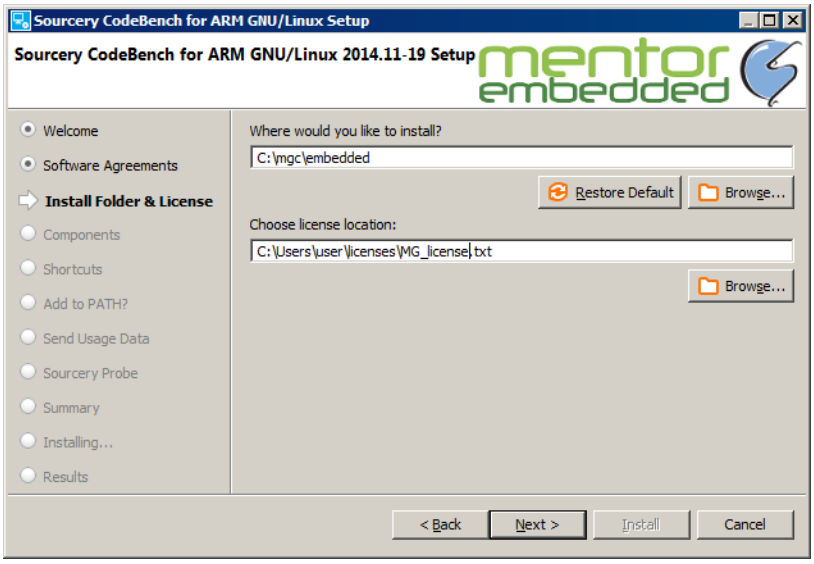
If you have an existing installation, select whether you want to install to a different location or upgrade the existing one.

After the installer starts, follow the on-screen dialogs to install Sourcery CodeBench Lite. The installer is intended to be self-explanatory and on most pages the defaults are appropriate.



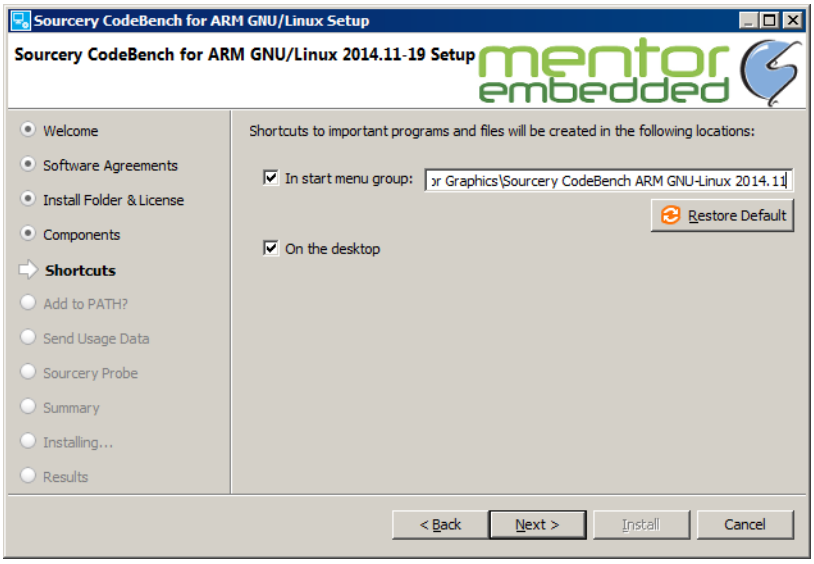
Running the Installer. The graphical installer guides you through the steps to install Sourcery CodeBench Lite.

You may want to change the install directory pathname and customize the shortcut installation.



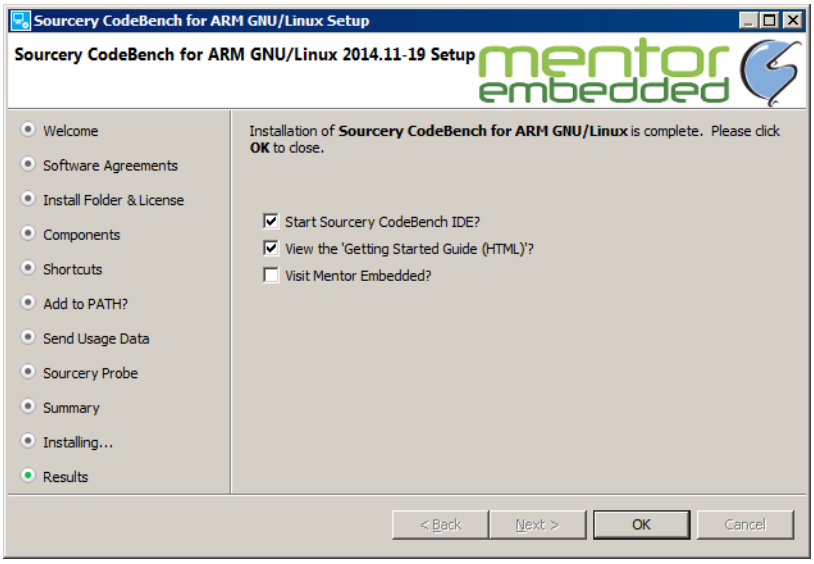
Choose Install Folder. Select the pathname to your install directory .

If you want to install add-ons, enter your Sourcery CodeBench Portal login information.



Choose Shortcut Folder. You can customize where the installer creates shortcuts for quick access to Sourcery CodeBench Lite.

When the installer has finished, it asks if you want to launch a viewer for the Getting Started guide, or visit Mentor® Embedded.



Install Complete. You should see a screen similar to this after a successful install.

If you prefer, you can run the installer in console mode rather than using the graphical interface. To do this, invoke the installer with the `-console` command-line option. For example:

```
> /path/to/package.exe -console
```

1.5.2. Using the Sourcery CodeBench Lite Installer on GNU/Linux Hosts

Start the graphical installer by invoking the executable shell script:

```
> /bin/sh ./path/to/package.bin
```

The installer gets extracted into the default temporary directory. If you want to specify the directory for installer extraction, change the `P2_INSTALLER_TEMP_PATH` variable.

After the installer starts, follow the on-screen dialogs to install Sourcery CodeBench Lite. For additional details on running the installer, see the discussion and screen shots in the Microsoft Windows section above.

If you prefer, or if your host system does not run the X Window System, you can run the installer in console mode rather than using the graphical interface. To do this, invoke the installer with the `-console` command-line option. For example:

```
> /bin/sh ./path/to/package.bin -console
```

1.5.3. Installing Sourcery CodeBench Lite from a Compressed Archive

You do not need to be a system administrator to install Sourcery CodeBench Lite from a compressed archive. You may install Sourcery CodeBench Lite using any user account and in any directory to which you have write access. This guide assumes that you have decided to install Sourcery CodeBench Lite in the `$HOME/CodeBench` subdirectory of your home directory and that the filename of the package you have downloaded is `/path/to/package.tar.bz2`. After installation the toolchain will be in `$HOME/CodeBench/sourceryg++-2016.05`.

First, uncompress the package file:

```
> bunzip2 /path/to/package.tar.bz2
```

Next, create the directory in which you wish to install the package:

```
> mkdir -p $HOME/CodeBench
```

Change to the installation directory:

```
> cd $HOME/CodeBench
```

Unpack the package:

```
> tar xf /path/to/package.tar
```

1.6. Installing Sourcery CodeBench Lite Updates

If you have already installed a Sourcery CodeBench Lite for MIPS ELF you must uninstall it before using the installer to unpack a new version in the same location.

If you are installing an update from a compressed archive, it is recommended that you remove any previous installation in the same location, or install in a different directory.

Note that the names of the Sourcery CodeBench commands for the MIPS ELF target all begin with `mips-sde-elf`. This means that you can install Sourcery CodeBench for multiple target systems in the same directory without conflicts.

1.7. Setting up the Environment

As with the installation process itself, the steps required to set up your environment depend on your host operating system.

1.7.1. Setting up the Environment on Microsoft Windows Hosts

1.7.1.1. Setting the `PATH`

The graphical installer for Sourcery CodeBench Lite does this setup for you, however it may not take effect until you next log in.

In order to use the Sourcery CodeBench tools from the command line, you should add them to your `PATH`. In the instructions that follow, replace *`installdir`* with the full pathname of your Sourcery CodeBench Lite installation directory, including the drive letter.

To set the `PATH` on a Microsoft Windows Vista system, use the following command in a `cmd.exe` shell:

```
> setx PATH "%PATH%;installdir\bin"
```

To set the `PATH` on a system running Microsoft Windows 7, from the desktop bring up the `Start` menu and right click on `Computer`. Select `Properties` and click on `Advanced system settings`. Go to the `Advanced` tab, then click on the `Environment Variables` button. Select the `PATH` variable and click `Edit`. Add the string `;`*`installdir`*`\bin` to the end, and click `OK`.

To set the `PATH` on a system running Microsoft Windows 8, navigate to the `Charms` menu. Click on `Search` then type `Control Panel`. Select `System` and click on `Advanced system`

`settings`. Go to the `Advanced` tab, then click on the `Environment Variables` button. Select the `PATH` variable and click `Edit`. Add the string `;installdir\bin` to the end, and click `OK`.

You can verify that your `PATH` is set up correctly by starting a new `cmd.exe` shell and running:

```
> mips-sde-elf-gcc -v
```

Verify that the last line of the output contains: `Sourcery CodeBench Lite 2016.05-7`.

1.7.1.2. Working with Cygwin

Sourcery CodeBench Lite does not require Cygwin or any other UNIX emulation environment. You can use Sourcery CodeBench command-line tools directly from the Windows command shell. You can also use Sourcery CodeBench from within the Cygwin environment, if you prefer.

The Cygwin emulation environment translates Windows path names into UNIX path names. For example, the Cygwin path `/home/user/hello.c` corresponds to the Windows path `c:\cygwin\home\user\hello.c`. Because Sourcery CodeBench is not a Cygwin application, it does not, by default, recognize Cygwin paths.

If you are using Sourcery CodeBench from Cygwin, you should set the `CYGPATH` environment variable. If this environment variable is set, Sourcery CodeBench Lite automatically translates Cygwin path names into Windows path names. To set this environment variable, type the following command in a Cygwin shell:

```
> export CYGPATH=cygpath
```

To resolve Cygwin path names, Sourcery CodeBench relies on the `cygpath` utility provided with Cygwin. You must provide Sourcery CodeBench with the full path to `cygpath` if `cygpath` is not in your `PATH`. For example:

```
> export CYGPATH=c:/cygwin/bin/cygpath
```

directs Sourcery CodeBench Lite to use `c:/cygwin/bin/cygpath` as the path conversion utility. The value of `CYGPATH` must be an ordinary Windows path, not a Cygwin path.

1.7.2. Setting up the Environment on GNU/Linux Hosts

The graphical installer for Sourcery CodeBench Lite does this setup for you, however it may not take effect until you next log in.

Before using Sourcery CodeBench Lite you should add it to your `PATH`. The command you must use varies with the particular command shell that you are using. If you are using the C Shell (`csh` or `tcsh`), use the command:

```
> setenv PATH installdir/bin:$PATH
```

If you are using Bourne Shell (`sh`), the Korn Shell (`ksh`), or another shell, use:

```
> PATH=installdir/bin:$PATH
> export PATH
```

If you are not sure which shell you are using, try both commands. In both cases, replace *`installdir`* with the full pathname of your Sourcery CodeBench Lite installation directory.

You may also wish to set the MANPATH environment variable so that you can access the Sourcery CodeBench manual pages, which provide additional information about using Sourcery CodeBench. To set the MANPATH environment variable, follow the same steps shown above, replacing PATH with MANPATH, and bin with share/doc/mips-mips-sde-elf/man.

You can test that your PATH is set up correctly by running the following command:

```
> mips-sde-elf-gcc -v
```

Verify that the last line of the output contains: Sourcery CodeBench Lite 2016.05-7.

1.8. Uninstalling Sourcery CodeBench Lite

The method used to uninstall Sourcery CodeBench Lite depends on the method you originally used to install it. If you have modified any files in the installation it is recommended that you back up these changes. The uninstall procedure removes all files and the installation directory, and it may remove files you have altered.

1.8.1. Using the Sourcery CodeBench Lite Uninstaller on Microsoft Windows

You should use the provided uninstaller to remove a Sourcery CodeBench Lite installation originally created by the graphical installer. Start the graphical uninstaller by invoking uninstall.exe located in your installation directory, or use the uninstall shortcut created during installation. After the uninstaller starts, follow the on-screen dialogs to uninstall Sourcery CodeBench Lite.

You can run the uninstaller in console mode, rather than using the graphical interface, by invoking uninstallc.exe found in your Sourcery CodeBench Lite installation directory with the -nosplash -install.console command-line option.

To uninstall third-party drivers bundled with Sourcery CodeBench Lite, first disconnect the associated hardware device. Then use Uninstall a program to remove the drivers separately. Depending on the device, you may need to reboot your computer to complete the driver uninstall.

1.8.2. Using the Sourcery CodeBench Lite Uninstaller on GNU/Linux

You should use the provided uninstaller to remove a Sourcery CodeBench Lite installation originally created by the executable installer script. Start the graphical uninstaller by invoking the uninstall binary located in your installation directory. After the uninstaller starts, follow the on-screen dialogs to uninstall Sourcery CodeBench Lite.

You can run the uninstaller in console mode, rather than using the graphical interface, by invoking the uninstall binary with the -install.console command-line option.

1.8.3. Uninstalling a Compressed Archive Installation

If you installed Sourcery CodeBench Lite from a .tar.bz2 file, you can uninstall it by manually deleting the installation directory created in the install procedure.

Chapter 2
Sourcery CodeBench Lite for MIPS ELF

This chapter contains information about features of Sourcery CodeBench Lite that are specific to MIPS ELF targets. You should read this chapter to learn how to best use Sourcery CodeBench Lite on your target system.

2.1. Included Components and Features

This section briefly lists the important components and features included in Sourcery CodeBench Lite for MIPS ELF, and tells you where you may find further information about these features.

Component	Version	Notes
GNU programming tools		
GNU Compiler Collection	5.3	Separate manual included.
GNU Binary Utilities	2.25.51	Includes assembler, linker, and other utilities. Separate manuals included.
Debugging support and simulators		
GNU Debugger	7.10.50	Separate manual included.
Sourcery CodeBench Debug Sprite	2016.05-7	See Chapter 5, “Sourcery CodeBench Debug Sprite”.
QEMU Emulator	2.2.0	See Section 3.3.1, “Connecting to the QEMU Emulator”.
Target libraries		
CodeSourcery Common Startup Code Sequence	2016.05-7	See Chapter 4, “CS3: The CodeSourcery Common Startup Code Sequence”.
Newlib C Library	2.2.0	Separate manuals included.
Other utilities		
GNU Make	N/A	Build support on Windows hosts.
GNU Core Utilities	N/A	Build support on Windows hosts.

2.2. Target Library Configurations

Sourcery CodeBench Lite for MIPS ELF includes the following library configuration.

MIPS32 revision 2 - Big-Endian, O32	
Command-line option(s):	default
Library subdirectory:	`./`

MIPS32 revision 2 - Little-Endian, O32	
Command-line option(s):	`-EL`
Library subdirectory:	`el/`

MIPS32 revision 2 - Big-Endian, O32, mips16	
Command-line option(s):	`-mips16`
Library subdirectory:	`mips16/`

MIPS32 revision 2 - Big-Endian, Soft-Float, O32	
Command-line option(s):	`-msoft-float`
Library subdirectory:	`sof/`

MIPS32 revision 2 - Big-Endian, O32, mips16, Soft-Float	
Command-line option(s):	`-mips16 -msoft-float`
Library subdirectory:	`mips16/sof/`

MIPS32 revision 2 - Little-Endian, O32, mips16	
Command-line option(s):	`-EL -mips16`
Library subdirectory:	`el/mips16/`

MIPS32 revision 2 - Little-Endian, O32, Soft-Float	
Command-line option(s):	`-EL -msoft-float`
Library subdirectory:	`el/sof/`

MIPS32 revision 2 - Little-Endian, O32, mips16, Soft-Float	
Command-line option(s):	`-EL -mips16 -msoft-float`
Library subdirectory:	`el/mips16/sof/`

MIPS32 revision 2 - Big-Endian, 2008 NaN, O32	
Command-line option(s):	`-mnan=2008`
Library subdirectory:	`nan2008/`

MIPS32 revision 2 - Little-Endian, 2008 NaN, O32	
Command-line option(s):	`-EL -mnan=2008`
Library subdirectory:	`el/nan2008/`

MIPS32 revision 2 - Little-Endian, O32, micromips, Soft-Float	
Command-line option(s):	`-EL -mmicromips -msoft-float`
Library subdirectory:	`el/micromips/sof/`

MIPS64 revision 2 - Big Endian, N64	
Command-line option(s):	`-mabi=64`
Library subdirectory:	`64/`

MIPS64 revision 2 - Big Endian, N64, Soft-Float	
Command-line option(s):	`-msoft-float -mabi=64`
Library subdirectory:	`sof/64/`

MIPS64 revision 2 - Little Endian, N64	
Command-line option(s):	`-EL -mabi=64`
Library subdirectory:	`el/64/`

MIPS64 revision 2 - Little Endian, N64, Soft-Float	
Command-line option(s):	`-EL -msoft-float -mabi=64`
Library subdirectory:	`el/sof/64/`

Sourcery CodeBench includes copies of run-time libraries that have been built with optimizations for different target architecture variants or other sets of build options. Each such set of libraries is referred to as a *multilib*. When you link a target application, Sourcery CodeBench selects the multilib matching the build options you have selected.

Sourcery CodeBench Lite's library support includes linker scripts that pull in appropriate CS3 startup code, as well as the libraries themselves. You can find these linker scripts in multilib-specific sub-directories of the `mips-sde-elf/lib` directory of your Sourcery CodeBench install.

2.3. CS3 Support

Sourcery CodeBench Lite includes CS3 linker scripts and initialization code to support three different classes of target configurations:

- Simulator targets, such as QEMU and MIPSsim, running under control of the debugger.
- Malta or SEAD-3 hardware targets running in a bare-metal configuration under control of the debugger.
- Malta or SEAD-3 hardware targets running under control of the YAMON boot monitor.

You must use the appropriate linker script to match your target, since the memory layouts and startup code sequences are different in each case. Refer to Chapter 4, “CS3: The CodeSourcery Common Startup Code Sequence” for details on the supported boards for this version of Sourcery CodeBench Lite.

For simulator and bare-metal targets, CS3 provides semihosted I/O via the debugger console on the host.

2.4. Using Sourcery CodeBench with MIPS Boards

The provided CS3 linker scripts for MIPS Malta and SEAD-3 boards (both bare-metal and YAMON profiles) assume a minimum amount of RAM is available on the target. Refer to the following table for the specific requirements. If your target board has less memory, you must adjust the memory layout used by the linker by specifying a custom linker script.

Board	**Memory Requirement**
Malta	128MB
SEAD-3 LX50	4MB
SEAD-3 LX110	128MB

Find the linker script for your selected profile, such as `mips-sde-elf/lib/malta-ram-hosted.ld`, in your Sourcery CodeBench Lite installation and copy it to your project working directory. In your local copy, find the `MEMORY` directive and edit the `LENGTH` expression to match the amount of memory available on your board. Then, use the full absolute pathname of your modified linker script with the `-T` command-line option when linking your program.

2.5. Using Sourcery CodeBench with YAMON

For YAMON targets, CS3 provides basic I/O services via the YAMON console. This section briefly covers how to load and run programs using YAMON.

To prepare an application to run from YAMON, you must first convert the executable file to SREC format. You can do this from the command line on your host system using the objcopy utility provided with Sourcery CodeBench Lite.

```
> mips-sde-elf-objcopy -O srec prog prog.srec
```

Next, use YAMON to load the SREC image file into RAM. For example, to load via TFTP, use a command similar to:

```
YAMON> load tftp://host/path/prog.srec
```

Then, start the program from the YAMON prompt:

```
YAMON> go .
```

For more detailed information about YAMON usage and features, refer to the *YAMON User's Manual*.

Warning

Using YAMON with 64-bit multilibs is not fully supported and may have issues. YAMON is usually built using the O32 ABI, which is not forward-compatible with the N64 ABI used in 64-bit multilibs.

2.6. Profiling Support

Sourcery CodeBench Lite includes CS3 support for code profiling on MIPS ELF targets using `gprof`. To enable profiling, compile your program with the `-pg` option. You must also build your program with a hosted linker script.

You can run a program built with profiling from the debugger the same as you would any other hosted application. While your program is running, profiling data is saved in buffers in the heap memory area on the target. This may affect the amount of memory available to your application, and it is also possible that the profiler itself may run out of memory. Profiling data is written to a file on the host (`gmon.out`) only when your application exits. Since many embedded applications are structured to run indefinitely rather than exit, you may need to add an explicit `exit` call in order to collect profiling data.

For instructions on using the `mips-sde-elf-gprof` utility to process the collected `gmon.out` data, refer to the GNU Profiler (gprof) manual included with Sourcery CodeBench Lite.

2.7. Using Flash Memory

Sourcery CodeBench Lite supports development and debugging of applications loaded into flash memory on MIPS ELF targets. There are three steps involved:

1. You must use an appropriate linker script that identifies the ROM memory region on your target board, and locates the program text within that region. Refer to Chapter 4, “CS3: The Code-

Sourcery Common Startup Code Sequence" for information about the boards supported by Sourcery CodeBench.

2. Next, load your program into the flash memory on your target board. You must use third-party tools to program the flash memory.

3. Finally, when debugging a program in flash memory, GDB must be told about the ROM region so that it knows where it must use hardware breakpoints to control program execution. If you are using the Sourcery CodeBench Debug Sprite to debug your program, the Sprite does this automatically, using the memory map provided in the board configuration file. Otherwise, you must provide this information explicitly.

   When using GDB from the command line, you can mark the flash memory as read-only by using the command:

   ```
   (gdb) mem start end ro
   ```

   where *start* and *end* define the address range of the read-only memory region.

In addition to GDB's automatic use of hardware breakpoints in the ROM region, you can also set hardware breakpoints explicitly from the debugger. However, on many targets the number of available hardware breakpoints is very small. Furthermore, GDB also uses hardware breakpoints internally to implement commands such as `step`, `next`, and `finish`. Thus the number of breakpoints you can explicitly set in ROM may be fewer than the number of hardware breakpoints supported by the target system.

Chapter 3
Using Sourcery CodeBench from the Command Line

This chapter demonstrates the use of Sourcery CodeBench Lite from the command line.

3.1. Building an Application

This chapter explains how to build an application with Sourcery CodeBench Lite using the command line. As elsewhere in this manual, this section assumes that your target system is mips-sde-elf, as indicated by the `mips-sde-elf` command prefix.

Using an editor (such as `notepad` on Microsoft Windows or `vi` on UNIX-like systems), create a file named `main.c` containing the following simple factorial program:

```
#include <stdio.h>

int factorial(int n) {
  if (n == 0)
    return 1;
  return n * factorial (n - 1);
}

int main () {
  int i;
  int n;
  for (i = 0; i < 10; ++i) {
    n = factorial (i);
    printf ("factorial(%d) = %d\n", i, n);
  }
  return 0;
}
```

Compile and link this program using the command:

```
> mips-sde-elf-gcc -o factorial main.c -T script
```

Sourcery CodeBench requires that you specify a linker script with the `-T` option to build applications for bare-board targets. Linker errors like `undefined reference to `read'` are a symptom of failing to use an appropriate linker script. Default linker scripts are provided in `mips-sde-elf/lib`. Refer to Chapter 4, “CS3: The CodeSourcery Common Startup Code Sequence” for information about the boards and linker scripts supported by Sourcery CodeBench Lite. You must also add the processor options for your board, as documented in that chapter, to your compile and link command lines.

There should be no output from the compiler. (If you are building a C++ application, instead of a C application, replace `mips-sde-elf-gcc` with `mips-sde-elf-g++`.)

3.2. Running Applications on the Target System

Consult your target board documentation for instructions on loading programs onto the target, and running them. Alternatively, you can use the Sourcery CodeBench Debug Sprite from within GDB to download and run programs on the target via a supported hardware debugging device.

3.3. Running Applications from GDB

You can run GDB, the GNU Debugger, on your host system to debug programs running remotely on a target board or system. GDB can also be used to run and debug programs with QEMU, a simulator that runs on your host system.

When starting GDB, give it the pathname to the program you want to debug as a command-line argument. For example, if you have built the factorial program as described in Section 3.1, “Building an Application”, enter:

```
> mips-sde-elf-gdb factorial
```

While this section explains the alternatives for using GDB to run and debug application programs, explaining the use of the GDB command-line interface is beyond the scope of this document. Please refer to the GDB manual for further instructions.

3.3.1. Connecting to the QEMU Emulator

Sourcery CodeBench Lite includes the QEMU emulator. This is a program which runs on your host computer and allows you to run and debug MIPS ELF applications without target hardware.

To start and connect to the emulator from within GDB, use this command:

```
(gdb) target qemu
```

This starts QEMU with the appropriate options to emulate a bare-board target and accept the connection from GDB.

You can optionally pass an argument to specify the CPU that QEMU should emulate:

```
(gdb) target qemu cpu
```

The default is `24Kf`. Additional supported CPU emulations include `4Kc`, `4Km`, `4KEcR1`, `4KEmR1`, `4KEc`, `4KEm`, `24Kc`, `34Kf`, `M14K`, and `M14Kc`.

In order to use QEMU as a debugging target, you must build your program with a QEMU linker script. Refer to Section 4.5, “Supported Boards for MIPS ELF” for details. You must also compile your code with options that are consistent with the processor you specify when invoking QEMU.

The version of QEMU included with Sourcery CodeBench Lite for MIPS ELF is configured to run in system emulation mode only, and other QEMU features not documented here are not supported in Sourcery CodeBench Lite. For additional information about QEMU, visit the QEMU web site[1].

3.3.2. Connecting to the Sourcery CodeBench Debug Sprite

The Sourcery CodeBench Debug Sprite is a program that runs on the host system to support hardware debugging devices. You can use the Debug Sprite to run and debug programs on a target board without an operating system, or to debug an operating system kernel. See Chapter 5, “Sourcery CodeBench Debug Sprite” for detailed information about the supported devices.

You can start the Sprite directly from within GDB:

```
(gdb) target remote | mips-sde-elf-sprite device-url board-file
```

Refer to Section 5.3, “Invoking Sourcery CodeBench Debug Sprite” for a full description of the Sprite arguments.

[1] http://fabrice.bellard.free.fr/qemu

3.3.3. Connecting to an External GDB Server

From within GDB, you can connect to a running `gdbserver` or other debugging stub that uses the GDB remote protocol using:

```
(gdb) target remote host:port
```

where *`host`* is the host name or IP address of the machine the stub is running on, and *`port`* is the port number it is listening on for TCP connections.

3.3.4. Loading and Running Applications

Connecting to a bare-metal target or simulator from GDB does not cause your program to be loaded into target memory. You must do this explicitly from GDB after you connect:

```
(gdb) load
```

Alternatively, you can use third-party tools to load your application into flash memory before starting GDB.

To begin execution of your application, you should generally use the `continue` command:

```
(gdb) continue
```

Chapter 4
CS3: The CodeSourcery Common Startup Code Sequence

CS3™ is CodeSourcery's low-level board support library. This chapter documents the boards supported by Sourcery CodeBench Lite and the compiler and linker options you need to use with them. It also explains how you can use and modify CS3-provided definitions for memory maps, system startup code and interrupt vectors in your own code.

Many developers turn to the GNU toolchain for its cross-platform consistency: having a single system support so many different processors and boards helps to limit risk and keep learning curves gentle. Historically, however, the GNU toolchain has lacked a consistent set of conventions for processor- and board-level initialization, language run-time setup, and interrupt and trap handler definition.

The CodeSourcery Common Startup Code Sequence (CS3) addresses this problem. For each supported system, CS3 provides a set of linker scripts describing the system's memory map, and a board support library providing generic reset, startup, and interrupt handlers. These scripts and libraries all follow a standard set of conventions across a range of processors and boards.

In addition to providing linker support, CS3's functionality is fully integrated with the Sourcery CodeBench Debug Sprite. For each supported board, CS3 provides the board file containing the memory map and initialization sequence required for debugging applications on the board via the Sprite, as documented in Section 5.7, “Supported Board Files”.

This chapter is organized in two parts. The first part explains CS3 concepts:

- Section 4.1, “Linker Scripts” provides basic information you need to know in order to select an appropriate CS3-provided linker script for your MIPS ELF board.
- CS3's program startup and termination model is discussed in Section 4.2, “Program Startup and Termination”.
- Section 4.3, “Memory Layout” discusses the mapping from program sections to memory regions. It also explains how you can refer to memory regions using CS3-provided symbolic names from C, assembly language, or the linker script, and customize placement of code or data in your program.

The second part provides details about the CS3 implementation for MIPS ELF:

- Section 4.5, “Supported Boards for MIPS ELF” lists the boards supported by CS3 for MIPS ELF, and the available linker scripts for them.

4.1. Linker Scripts

When you build programs for MIPS ELF targets, you must use a linker script. The linker script serves several purposes:

- It determines the memory addresses for placement of code and data sections.
- It defines symbolic names for memory regions present on the board, which you can use programmatically within your code.
- It provides appropriate program startup and termination code, and causes the linker to pull in any low-level board support libraries that are required to run code on the target.
- It optionally provides a *hosting* library for basic I/O functionality.
- It provides a default interrupt vector appropriate for the target processor.

When invoking the Sourcery CodeBench linker from the command line, you must explicitly supply a linker script using the `-T` option; otherwise a link error results.

CS3 may provide multiple linker scripts for different configurations using the same board. For example, on some boards CS3 may support running the program from either RAM or ROM (flash). Some CS3 link configurations are also designed to co-exist with, or be run from, a boot monitor on

the target board. Simulator targets typically require different startup code configurations than hardware targets. In CS3 terminology, each of these different configurations is referred to as a *profile*.

The remainder of this section discusses profile and hosting selection considerations in more detail. You can find the full list of supported boards and linker scripts included in this release of Sourcery CodeBench Lite in Section 4.5, “Supported Boards for MIPS ELF”.

4.1.1. Program and Data Placement

Many boards have both RAM and ROM (flash) memory devices. CS3 provides distinct linker scripts to place the application either entirely in RAM, or to place code and read-only data in ROM.

Some boards have very small amounts of RAM memory. If you use large library functions (such as `printf` and `malloc`), you may overflow the available memory. You may need to use the ROM-based profile for such programs, so that the program itself is stored in ROM. You may be able to reduce the total amount of memory used by your program by replacing portions of the Sourcery CodeBench runtime library and/or startup code.

4.1.2. Hosting and Semihosting

CS3 is designed to support boards without an operating system. To allow functions like `open` and `write` to work without operating system support, a *semihosting* feature is supported, in conjunction with the debugger.

With semihosting enabled, these system calls are translated into equivalent function calls on your host system. You can only use these function calls while connected to the debugger; if you try to use them when disconnected from the debugger, you will get a hardware exception.

Semihosting requires support from the remote GDB debugging stub or agent, as well as the debugger itself. The Sourcery CodeBench Debug Sprite implements semihosting for all supported devices. Semihosting is also supported by the QEMU Emulator included with Sourcery CodeBench Lite. However, semihosting may not be supported by debugging stubs provided by third parties. If you are using a debug device that communicates with GDB using the GDB remote protocol, check the documentation for your device to see whether semihosting is supported.

A good use of semihosting is to display debugging messages. For example, this program prints a message on the debugger console on the host:

```
#include <unistd.h>

int main () {
  write (STDERR_FILENO, "Hello, world!\n", 14);
  return 0;
}
```

The hosted CS3 linker scripts provide the semihosting support, and as such programs linked with them may only be run with the debugger. For production code, or programs where memory usage is tightly constrained, use the unhosted CS3 linker scripts instead. These scripts provide stub versions of the system calls, which return an appropriate error value in `errno`. If such a stub system call is required in the executable, the linker also produces a warning. Such a warning may indicate that you have left debugging code active, or that your program contains unused code.

As an alternative to semihosting via the debugger, some targets supported by CS3 can run a boot monitor that provides console I/O services and other basic system calls. CS3 can also provide hosting

via these facilities; where a boot monitor is supported, this is noted in the board tables below. Unlike semihosting, hosting via the boot monitor can be used when running programs outside of the debugger.

4.1.3. Specifying a Linker Script

When using Sourcery CodeBench from the command line or from a `Makefile`, you must add `-T` *`script`* to your linking command, where *`script`* is the appropriate linker script. For example, to target MIPS Malta boards, you could link with `-T malta-yamon.ld`.

4.2. Program Startup and Termination

This section documents CS3's model for target initialization prior to invoking the `main` function of your program, and aspects of program termination that are left unspecified in the C and C++ standards. It explains how you can customize or override the default behavior for your application.

CS3 divides the startup sequence into three phases:

- The *hard reset phase* (`__cs3_reset`) includes actions such as initializing the memory controller and setting up the memory map.
- The *assembly initialization phase* (`__cs3_start_asm`) prepares the stack to run C code, and jumps to the C initialization function.
- The *C initialization phase* (`__cs3_start_c`) is responsible for initializing the data areas, running constructors for statically-allocated objects, and calling `main`.

The hard reset and assembly initialization phases are necessarily written in assembly language; at reset, there may not yet be stack to hold compiler temporaries, or perhaps even any RAM accessible to hold the stack. These phases do the minimum necessary to prepare the environment for running simple C code. Then, the code for the final phase may be written in C; CS3 leaves as much as possible to be done at this point.

The CodeSourcery board support library provides default code for all three phases. The hard reset phase is implemented by board- and profile-specific code. The assembly initialization phase is implemented by profile-specific code. The C initialization phase is implemented by generic code.

4.2.1. The Hard Reset Phase

This phase, which begins at `__cs3_reset`, is responsible for initializing board-specific registers, such as memory base registers and DRAM controllers, or scanning memory to check the available size. It is written in assembler and ends with a jump to `__cs3_start_asm`, which is where the assembly initialization phase begins.

The hard reset code is in a section named `.cs3.reset`. CS3 linker scripts define `__cs3_reset` as an alias for a board- and profile-specific entry point. You may override the CS3-provided reset code by defining your own `__cs3_reset` entry point in the `.cs3.reset` section.

Program execution always begins at `__cs3_reset`, whether the program is started from the reset vector, the debugger, or a boot monitor. However, the `__cs3_reset` code linked into the application is typically non-empty only for ROM-based profiles. For example, in a RAM-based profile, resetting the memory controllers would overwrite the code being executed.

When using the Sourcery CodeBench Debug Sprite, the Sprite is responsible for carrying out the hard reset actions before the program is loaded onto the target. This is performed prior to execution of both RAM- and ROM-profile applications from the debugger. Thus, when debugging a ROM-

profile application, hard reset is actually performed twice — once by the Sprite, and once by the application itself.

4.2.2. The Assembly Initialization Phase

This phase is responsible for initializing the stack pointer and creating an initial stack frame. The symbol `__cs3_start_asm` marks the entry point of the assembly initialization code. The assembly initialization phase ends with a call or jump to `__cs3_start_c`.

The assembly initialization phase is profile-specific. For example, while bare-board applications typically must initialize the stack themselves, CS3 also supports boot-monitor profiles where the stack is initialized by the boot monitor before it launches the application. Likewise, some simulators automatically initialize the stack pointer and initial stack frame on startup, while others require a supervisory operation on startup to determine the amount of available memory. Each of these scenarios requires different assembly initialization behavior.

Note that on bare-board targets setting the stack pointer explicitly in the assembly initialization phase is required even if the processor itself initializes the stack pointer automatically on reset. This is to support running programs from the debugger as well as from processor reset.

For backwards compatibility with previous versions of CS3, on RAM and ROM profiles the symbol `__cs3_start_asm` is actually an alias for a symbol named `_start`. However, referencing or defining `_start` directly is now deprecated.

The value of the symbol `__cs3_stack` provides the initial value of the stack pointer for profiles that must set it explicitly. The CodeSourcery linker scripts provide a default value for this symbol, which you may override by defining `__cs3_stack` yourself. See Section 4.3.3, “Heap and Stack Placement” for an example of a custom stack.

The initial stack frame is created for the use of ordinary C and C++ calling conventions. The stack should be initialized so that backtraces stop cleanly at this point; this might entail zeroing a dynamic link pointer, or providing hand-written DWARF call frame information.

The last action of the assembly initialization phase is to call the C function `__cs3_start_c`. This function never returns, and `__cs3_start_asm` need not be prepared to handle a return from it.

As with the hard reset code, the CodeSourcery board support library provides reasonable default assembly initialization code. However, you may provide your own code by providing a definition for `__cs3_start_asm`, either in an object file or a library.

4.2.3. The C Initialization Phase

Finally, C code can be executed. The C startup function is declared as follows:

```
void __cs3_start_c (void) __attribute__ ((noreturn));
```

This function performs the following steps:

- Initialize all `.data`-like sections by copying their contents. For example, ROM-profile linker scripts use this mechanism to initialize writable data in RAM from the read-only data program image.
- Clear all `.bss`-like sections.
- Run constructors for statically-allocated objects, recorded using whatever conventions are usual for C++ on the target architecture.

CS3 reserves priorities from 0 to 100 for use by initialization code. You can handle tasks like enabling interrupts, initializing coprocessors, pointing control registers at interrupt vectors, and so on by defining constructors with appropriate priorities.

- Call `main` as appropriate.
- Call `exit`, if it is available.

As with the hard reset and assembly initialization code, the CodeSourcery board support library provides a reasonable definition for the `__cs3_start_c` function. You may override this by providing a definition for `__cs3_start_c`, either in an object file or in a library.

4.2.4. Arguments to `main`

The CodeSourcery-provided definition of `__cs3_start_c` can pass command-line arguments to `main` using the normal C `argc` and `argv` mechanism if the board support package provides corresponding definitions for `__cs3_argc` and `__cs3_argv`. For example:

```
int __cs3_argc;
char **__cs3_argv;
```

These variables should be initialized using a constructor function, which is run by `__cs3_start_c` after it initializes the data segment. Use the `constructor` attribute on the function definition:

```
__attribute__((constructor))
void __cs3_init_args (void) {
   __cs3_argc = ...;
   __cs3_argv = ...;
}
```

The constructor function must be named `__cs3_init_args`, since some profiles provide a default definition of this function.

If definitions of `__cs3_argc` and `__cs3_argv` are not provided, then the default `__cs3_start_c` function invokes `main` with zero as the `argc` argument and a null pointer as `argv`.

4.2.5. Program Termination

A program running on an embedded system is usually designed never to exit — it runs until the system is powered down. The C and C++ standards leave it unspecified as to whether `exit` is called at program termination. If the program never exits, then there is no reason to include `exit`, facilities to run functions registered with `atexit`, or global destructors. This code would never be run and would therefore just waste space in the application.

The CS3 startup code, by itself, does not cause `exit` to be present in the application. It dynamically checks whether `exit` is present, and only calls it if it is. If you require `exit` to be present, either refer to it within your application, or add `-Wl,-u,exit` to the linking command line.

Similarly, code to register global destructors is only invoked when `atexit` is already in the executable; CS3, by itself, does not cause `atexit` to be present. If you require `atexit`, either refer to it within your application, or add `-Wl,-u,atexit` to the linking command line.

4.3. Memory Layout

Boards supported by CS3 can have multiple banks or regions of memory with different characteristics. This section describes how program sections are mapped onto memory regions, and how you can use these CS3 features to customize placement of your program's code or data in memory. CS3 also provides a uniform set of symbolic names for each region, allowing you to programmatically refer to each region's address range from C or assembly language as well as from the linker script.

4.3.1. Memory Regions and Program Sections

The regions that are available on a particular board are listed in the table for that board in Section 4.5, “Supported Boards for MIPS ELF”, below. There are two kinds of regions: those documented as "Memory regions", which are general-purpose memory banks that can be used for program or data storage; and those documented as "Other regions", which typically correspond to memory-mapped control registers or other special-purpose storage.

CS3 supports boards that include both `ram` and `rom` memory regions. The `ram` region holds the `.data` and `.bss` sections, and the `.text` section in RAM profiles. In ROM profiles, the `rom` region holds the `.text` section and initialization values for the writable data sections.

In addition, all regions documented as "Memory regions" correspond to similarly-named program sections. For example, the linker script assigns the `.ram` section to the `ram` region.

More generally, for a memory region named *R*, CS3 linker scripts define a section named `.`*R*, which may contain initialized data or code. There is also a section named `.bss.`*R* for zero-initialized data (BSS), which is placed after the initialized data section for this region.

You can explicitly locate data or code in a section corresponding to a particular memory region using section attributes in your source C or C++ code. Section attributes are especially useful on code compiled for boards that include special memory banks, such as a fast on-chip cache memory, in addition to the default `ram` and/or `rom` regions. CS3's start-up code arranges for additional data-like sections to be initialized in the same way as the default `.data` section.

As an example to illustrate the attribute syntax, you can put a variable `v` in the `.ram` section using:

```
int v __attribute__ ((section (".ram")));
```

To declare a function `f` in this section, use:

```
int f (void) __attribute__ ((section (".ram"))) {...}
```

For more information about attribute syntax, see the GCC manual.

In addition to the `.`*R* and `.bss.`*R* sections, CS3 places a `.cs3.region-head.`*R* section at the beginning of each region *R*. Explicitly placing data in `.cs3.region-head.`*R* sections is discouraged, because CS3 itself may want to place items (like interrupt vector tables) at these locations. If there is a conflict, CS3 raises an error at link time.

Regions documented as "Other regions" in the tables in Section 4.5, “Supported Boards for MIPS ELF” do not have corresponding program sections. Typically, these regions contain memory-mapped control and I/O registers and cannot be used for general data or program storage. If your program needs to manipulate data in these regions, you can use the CS3 memory map access interface declared in `cs3.h`, as described in Section 4.3.2, “Programmatic Access to the CS3 Memory Map”.

Memory maps for boards supported by Sourcery CodeBench Lite for MIPS ELF are documented in the linker scripts in the `mips-sde-elf/lib/` subdirectory of your Sourcery CodeBench installation directory.

4.3.2. Programmatic Access to the CS3 Memory Map

CS3 makes C declarations describing the memory regions on the target board available to your program via the header file `cs3.h`, which you can find in the `mips-sde-elf/include` directory within your install.

For each region named *R*, `cs3.h` declares a byte array variable `__cs3_region_start_`*R* at the region's start address, and a `size_t` variable `__cs3_region_size_`*R* to represent the total size of the region. These symbols are defined by the linker script and so may also be referenced from assembly language. Note that all regions are aligned on eight-byte boundaries and sizes are also multiples of eight bytes.

For memory regions that can correspond to program sections (as described in Section 4.3.1, "Memory Regions and Program Sections"), there are additional symbols `__cs3_region_init_`*R* and `__cs3_region_init_size_`*R* that describe constant data used to initialize the region. During the C initialization phase (Section 4.2, "Program Startup and Termination"), this data is copied into the lower part of the memory region. The symbol `__cs3_region_zero_size_`*R* represents the size of the zero-initialized `.bss.`*R* section following the initialized data. Any of these identifiers may actually be defined as a preprocessor macro that expands to an expression of the appropriate type and value.

To perform the memory region initializations during startup, CS3 internally uses the array variable `__cs3_regions`, which contains descriptors for all of the writable (RAM) memory regions. These descriptors are also exposed in `cs3.h`; refer to the header file for details.

4.3.3. Heap and Stack Placement

CS3 linker scripts provide default placement of the heap and stack in the RAM region. However, you can override the defaults by providing your own definitions of the associated CS3 variables. For example, you may put the heap and/or stack in some other memory region.

Heap placement is controlled by defining the symbol `__cs3_heap_start` at the beginning of the heap, and either the symbol `__cs3_heap_end` or the pointer variable `__cs3_heap_limit` to mark the end of the heap. For example, this fragment of C code places the heap in a region named `extsram`:

```
#define HEAPSIZE ...  /* However big you want to make it. */
unsigned char __cs3_heap_start[HEAPSIZE]
    __attribute__ ((section (".bss.extsram"), aligned(8)));
unsigned char *__cs3_heap_limit = __cs3_heap_start + HEAPSIZE;
```

The default initial stack pointer for bare-metal profiles is given by the symbol `__cs3_stack`, and the stack grows downward from this address. Stack initialization is discussed in more detail in Section 4.2.2, "The Assembly Initialization Phase".

You can find C declarations for the CS3 heap and stack symbols in the header file `cs3.h`.

The `cs3.h` header file also defines a macro for creating a custom stack. The custom stack is created as a block of RAM in the zero-initialized data section (BSS). The specified size must be a compile-time constant. To account for alignment, the final size of the stack may be a few bytes less than the

requested size. The symbol `__cs3_stack` is initialized to point to the last extent of the stack block, and is 16-byte aligned. For example, the following fragment of C code creates a stack of 8192 bytes:

```
#include <cs3.h>

CS3_STACK(2 * 4096);
```

As indicated in Section 4.2.2, “The Assembly Initialization Phase”, there are cases where a boot monitor or simulator overrides a custom stack.

4.4. Interrupt Vectors and Handlers

CS3 provides standard handlers for interrupts, exceptions and traps, but also allows you to define your own handlers as needed. In this section, we use the term *interrupt* as a generic term for this entire class of events.

Different processors handle interrupts in various ways, but there are three general approaches:

- Some processors fetch an address from an array indexed by the interrupt number, and jump to that address. We call these *address vector* processors.
- Others multiply the interrupt number by some constant factor, add a base address, and jump directly to that address. Here, the interrupt vector consists of blocks of code, so we call these *code vector* processors.
- Still other processors use a more complicated descriptor mechanism for the interrupt table.

MIPS processors use the code vector model. The remainder of this section assumes that you have some understanding of the specific requirements for your target; refer to the architecture manuals if necessary.

4.4.1. MIPS ELF Interrupt Vector Implementation

On MIPS ELF targets, CS3 provides interrupt and exception handling support using the MIPS SDE library interface, which is integrated with the exception support provided by the YAMON boot monitor. The interfaces are modelled on the POSIX signal handling mechanism and are declared in the C header file `mips/xcpt.h`.

4.4.2. Writing Interrupt Handlers

Interrupt handlers typically require special call/return and register usage conventions that are target-specific and beyond the scope of this document. In many cases, normal C functions cannot be used as interrupt handlers.

As an alternative to writing interrupt handlers in assembly language, on MIPS targets they may be written in C using the `interrupt` attribute. This tells the compiler to generate appropriate function entry and exit sequences for an interrupt handler. There are additional MIPS-specific attributes you can specify to modify the behavior of the interrupt handler. Refer to the GCC manual for more details about attribute syntax and usage.

4.5. Supported Boards for MIPS ELF

CS3 provides support for the following boards on MIPS ELF targets.

MIPS Malta		
Processor name:	unspecified	
Processor options:	none	
Memory regions:	`ram`	
Linker scripts:	RAM Hosted	`malta-ram-hosted.ld`
	RAM Unhosted	`malta-ram.ld`
	YAMON	`malta-yamon.ld`

MIPS Malta 64-bit		
Processor name:	unspecified	
Processor options:	none	
Memory regions:	`ram`	
Linker scripts:	RAM Hosted	`malta64-ram-hosted.ld`
	RAM Unhosted	`malta64-ram.ld`
	YAMON	`malta64-yamon.ld`

MIPS SEAD-3 LX110		
Processor name:	unspecified	
Processor options:	none	
Memory regions:	`ram`	
Linker scripts:	RAM Hosted	`sead3-lx110-ram-hosted.ld`
	RAM Unhosted	`sead3-lx110-ram.ld`
	YAMON	`sead3-lx110-yamon.ld`

MIPS SEAD-3 LX50		
Processor name:	unspecified	
Processor options:	none	
Memory regions:	`ram`, `isram` (64K Instruction SRAM), `dsram` (64K Data SRAM)	
Linker scripts:	RAM Hosted	`sead3-lx50-ram-hosted.ld`
	RAM Unhosted	`sead3-lx50-ram.ld`
	Dual SRAM Hosted	`sead3-lx50-dual-sram-hosted.ld`
	Dual SRAM Unhosted	`sead3-lx50-dual-sram.ld`
	YAMON	`sead3-lx50-yamon.ld`
	YAMON Dual SRAM	`sead3-lx50-yamon-dual-sram.ld`

MIPS Simulator		
Processor name:	unspecified	
Processor options:	none	
Memory regions:	`ram`	
Linker scripts:	Simulator Hosted	`mipssim-hosted.ld`
	Simulator Unhosted	`mipssim.ld`

QEMU Simulator		
Processor name:	unspecified	
Processor options:	none	
Memory regions:	`ram`	
Linker scripts:	Simulator Hosted	`qemu-hosted.ld`
	Simulator Unhosted	`qemu.ld`

QEMU Simulator 64-bit		
Processor name:	unspecified	
Processor options:	none	
Memory regions:	`ram`	
Linker scripts:	Simulator Hosted	`qemu64-hosted.ld`
	Simulator Unhosted	`qemu64.ld`

Chapter 5
Sourcery CodeBench Debug Sprite

This chapter describes the use of the Sourcery CodeBench Debug Sprite for remote debugging. The Sprite allows you to debug programs running on a bare board without an operating system. This chapter includes information about the debugging devices and boards supported by the Sprite for MIPS ELF.

Sourcery CodeBench Lite contains the Sourcery CodeBench Debug Sprite for MIPS ELF. This Sprite is provided to allow debugging of programs running on a bare board. You can use the Sprite to debug a program when there is no operating system on the board, or for debugging the operating system itself. If the board is running an operating system, and you wish to debug a program running on that OS, you should use the facilities provided by the OS itself (for instance, using `gdbserver`).

The Sprite acts as an interface between GDB and external debug devices and libraries. Refer to Section 5.3, "Invoking Sourcery CodeBench Debug Sprite" for information about the specific devices supported by this version of Sourcery CodeBench Lite.

Important

The Sourcery CodeBench Debug Sprite is not part of the GNU Debugger and is not free or open-source software. You may use the Sourcery CodeBench Debug Sprite only with the GNU Debugger. You may not distribute the Sourcery CodeBench Debug Sprite to any third party.

5.1. Probing for Debug Devices

Before running the Sourcery CodeBench Debug Sprite for the first time, or when attaching new debug devices to your host system, it is helpful to verify that the Sourcery CodeBench Debug Sprite recognizes your debug hardware. From the command line, invoke the Sprite with the `-i` option:

```
> mips-sde-elf-sprite -i
```

This prints out a list of supported device types. For devices that can be autodetected, it additionally probes for and prints out a list of attached devices. For instance:

```
Sourcery CodeBench Debug Sprite for MIPS (Sourcery CodeBench Lite \
2016.05-7)
mdi: [lib=<file>&cfg=<file>&rst=<n>] MDI device
  mdi:///23/1 - 24KE     (Instruction)/24KE LE
  mdi:///23/2 - 24KE     (Instruction)/24KE BE
  mdi:///24/1 - 24KE     (Cycle)/24KE LE
  mdi:///24/2 - 24KE     (Cycle)/24KE BE
  mdi:///$Target/$Device - Generic MDI target/device
```

This shows that MDI (Microprocessor Debug Interface) devices are supported. Four MIPSsim devices have been autodetected. Note that additional configuration steps for the MDI library are required to allow the Sprite to autodetect devices; see Section 5.5, "MDI Devices".

5.2. Debug Sprite Example

Start by compiling and linking a simple test program for your target board, following the instructions in Chapter 3, "Using Sourcery CodeBench from the Command Line". Use the `-g` option to tell the compiler to generate debugging information.

For example, to build the `factorial` program to run on MIPSsim, use:

```
> mips-sde-elf-gcc -g -Tmipssim-hosted.ld  main.c -o factorial
```

Next start the debugger on your host system:

```
> mips-sde-elf-gdb factorial
```

To connect GDB to the MDI target, use a command similar to:

```
(gdb) target remote | mips-sde-elf-sprite mdi:///23/2 mipssim
```

Refer to Section 5.5, “MDI Devices” for additional set-up required to use the Sprite with MDI devices.

The Sprite prints some status messages as it connects to your debug device and target board. If the connection is successful, you should see output similar to:

```
mips-sde-elf-sprite:Target reset
0x00008936 in ?? ()
(gdb)
```

Next, use GDB to load your program onto the target board.

```
(gdb) load
```

At this point you can use GDB to control the execution of your program as required. For example:

```
(gdb) break main
(gdb) continue
```

5.3. Invoking Sourcery CodeBench Debug Sprite

The Debug Sprite is invoked as follows:

```
> mips-sde-elf-sprite [options] device-url board-file
```

The *device-url* specifies the debug device to use to communicate with the board. It follows the standard format:

```
scheme:scheme-specific-part[?device-options]
```

Most device URL schemes also follow the regular format:

```
scheme:[//hostname:[port]]/path[?device-options]
```

The meanings of *hostname*, *port*, *path* and *device-options* parts depend on the *scheme* and are described below. The following schemes are supported in Sourcery CodeBench Lite for MIPS ELF:

`mdi` Use a Microprocessor Debug Interface (MDI) debugging device. Refer to Section 5.5, “MDI Devices”.

The optional `?`*device-options* portion is allowed in all schemes. These allow additional device-specific options of the form *name*`=`*value*. Multiple options are concatenated using `&`.

The *board-file* specifies an XML file that describes how to initialize the target board, as well as other properties of the board used by the debugger. If *board-file* refers to a file (via a relative or absolute pathname), it is read. Otherwise, *board-file* can be a board name, and the toolchain's board directory is searched for a matching file. See Section 5.7, “Supported Board Files” for the list of supported boards, or invoke the Sprite with the `-b` option to list the available board files. You can also write a custom board file; see Section 5.8, “Board File Syntax” for more information about the file format.

Both the *device-url* and *board-file* command-line arguments are required to correctly connect the Sprite to a target board.

5.4. Sourcery CodeBench Debug Sprite Options

The following command-line options are supported by the Sourcery CodeBench Debug Sprite:

`-a` Attach to a process already running on the target. Without this option, the default behavior is to reset the target on the initial connection, in preparation for loading a new program from the debugger.

`-b` Print a list of *board-file* files in the board config directory.

`-h` Print a list of options and their meanings. A list of *device-url* syntaxes is also shown.

`-i` Print a list of the accessible devices. If a *device-url* is also specified, only devices for that device type are scanned. Each supported device type is listed along with the options that can be appended to the *device-url*. For each discovered device, the *device-url* is printed along with a description of that device.

`-l` *[host]:port* Specify the host address and port number to listen for a GDB connection. If this option is not given, the Debug Sprite communicates with GDB using stdin and stdout. If you start the Sprite from within GDB using the `target remote | mips-sde-elf-sprite ...` command, you do not need this option.

`-m` Listen for multiple sequential connections. Normally the Debug Sprite terminates after the first connection from GDB terminates. This option instead makes it listen for a subsequent connection. To terminate the Sprite, open a connection and send the string `END\n`.

`-q` Do not print any messages.

`-v` Print additional messages.

If any of `-b`, `-i` or `-h` are given, the Debug Sprite terminates after providing the information rather than waiting for a debugger connection.

5.5. MDI Devices

The Sourcery CodeBench Debug Sprite for MIPS supports MDI (Microprocessor Debug Interface) devices. Each MDI device is identified by a target number and device number; these form the *path* part of the device URL, and the *hostname* and *port* must be empty or omitted. Thus, the *device-url* has the form:

```
mdi:///targetnum/devicenum[?device-options]
```

You can also use the environment variables `GDBMDITARGET` and `GDBMDIDEVICE` to provide defaults for the *targetnum* and *devicenum*.

The following *device-options* are permitted:

`lib=`*`filename`*

This option specifies the MDI library to load. It is equivalent to setting the `GDBMDILIB` environment variable.

`cfg=`*`filename`*

Some MDI target libraries, such as MIPSsim, require a configuration file. (This is distinct from the Sprite's own *`boardfile`*.) You can use this option to specify the file. It is equivalent to setting the `GDBMIPSSIMCONFIG` environment variable.

`rst=`*`seconds`*

This option can be used to specify a delay after the target is reset by the Sprite. If the value of *`seconds`* is greater than zero, then execution is resumed for the specified number of seconds; this can be used to allow power-on firmware to initialize the memory controller and peripherals. Then the target is halted again and queried for configuration.

If the value of *`seconds`* is `-1`, then the target is queried immediately without reset. This is the same effect as passing the `-a` command-line option to the Sprite, which allows the Sprite to attach to a running program.

This option is equivalent to setting the `GDBMDICONNRST` environment variable. If neither the option nor the environment variable are provided, the default is to reset the target and query it immediately unless the `-a` option is specified.

`group=`*`/targetn/devicen`*

This option may be specified multiple times and is cumulative. Each of the specified devices is opened and queried and they are all treated as threads of execution, subject to being enabled or active; if a device is disabled or has no active thread contexts associated with it, it is not visible to GDB but is still under control of the Sprite in case its state changes. This option cannot be used in combination with the `team=` option.

`team=`*`/targetn/devicen`*

This option may be specified multiple times and is cumulative. The specified devices are not opened, but are associated with the base device by means of the MDI team mechanism for the purpose of synchronization. The specified devices may still be opened and controlled by another debugger (such as another instance of the Debug Sprite) independently. This option cannot be used in combination with the `group=` option.

Before you can connect to a target using the MDI API, you must tell the Debug Sprite which shared library or DLL to load for your simulator or device. On Linux hosts you should add the directory containing the shared library files to your `LD_LIBRARY_PATH` environment variable. On Windows hosts, add the directory containing the DLLs to your `PATH` environment variable. Then, either set the environment variable `GDBMDILIB` to the base name of the MDI library before starting the Debug Sprite, or use the `lib=` device option to specify the library to load.

Similarly, the `-i` command-line option can only probe for devices if you have set the `PATH` or `LD_LIBRARY_PATH` environment variable appropriately, and specify an MDI library using either the `GDBMDILIB` environment variable or the `lib=` device option. Otherwise, it reports only the generic *`device-url`* syntax.

For example, to use an FS2 probe on a Windows host to debug a MIPS Malta board, first add the directory containing the MDI DLLs to your `PATH`. Then you can invoke the Sprite from GDB using a command line similar to:

```
(gdb) target remote | mips-sde-elf-sprite \
'mdi:///2/2?lib=jnetfs2mdilib.dll&rst=7' malta
```

The quotes are required to prevent special characters in the *`device-url`* from being interpreted by the shell.

In the above command, the `rst=7` option provides for a sufficient delay for the board's reset code to execute on connection. Since this takes several seconds, GDB may time out waiting for the Sprite to respond. You can prevent this by issuing this command before you connect to the Sprite:

```
(gdb) set remotetimeout 10
```

To use the Sprite with MIPSsim, a configuration file is required. The configuration files provided with the MIPSsim distribution are intended for use with standalone execution from the command line, rather than running the program from the debugger. So, make a copy and comment out the `APP_FILE` setting. It is also recommended that you comment out `TRACE_FILE` as well, since the trace files can be very large.

To connect to MIPSsim using the Sprite on a Linux host, first set your `LD_LIBRARY_PATH` and `GDBMDILIB` as described above. You can run the Sprite from the shell to probe for devices to verify that your setup is correct:

```
> mips-sde-elf-sprite -i
```

Then, from GDB, use a command similar to:

```
(gdb) target remote | mips-sde-elf-sprite \
'mdi:///23/2?cfg=24KE.cfg&rst=-1' mipssim
```

Fill in your target and device numbers as reported by the probe output, and the full pathname to your configuration file. The `rst=-1` option is required, as MIPSsim does not support reset.

This section describes only the basic MDI usage; refer to the documentation for your MDI simulator or debug device for details specific to that target. Note, in particular, that some MDI targets may require you to set up a license in addition to the steps given here.

5.6. Debugging a Remote Board

You can run the Sourcery CodeBench Debug Sprite on a different machine from the one on which GDB is running. For example, if your board is connected to a machine in your lab, you can run the debugger on your laptop and connect to the remote board. The Sourcery CodeBench Debug Sprite must run on the machine that is connected to the target board. You must have Sourcery CodeBench installed on both machines.

To use this mode, you must start the Sprite with the `-l` option and specify the port on which you want it to listen. For example:

```
> mips-sde-elf-sprite -l :10000 device-url board-file
```

starts the Sprite listening on port 10000.

When running GDB from the command line, use the following command to connect GDB to the remote Sprite:

```
(gdb) target remote host:10000
```

where *host* is the name of the remote machine. After this, debugging is just as if you are debugging a target board connected to your host machine.

For more detailed instructions on using the Sourcery CodeBench Debug Sprite in this way, please refer to the Sourcery CodeBench Knowledge Base[1].

5.7. Supported Board Files

The Sourcery CodeBench Debug Sprite for MIPS ELF includes support for the following target boards. Specify the appropriate *board-file* as an argument when invoking the Sprite from the command line.

Board	**Config**
MIPS Malta	`malta`
MIPS Malta 64-bit	`malta64`
MIPS SEAD-3 LX110	`sead3-lx110`
MIPS SEAD-3 LX50	`sead3-lx50`
MIPS Simulator	`mipssim`

5.8. Board File Syntax

The *board-file* can be a user-written XML file to describe a non-standard board. The Sourcery CodeBench Debug Sprite searches for board files in the `mips-sde-elf/lib/boards` directory in the installation. Refer to the files in that directory for examples.

The file's DTD is:

```
<!-- Board description files

     Copyright (c) 2007-2012 Mentor Graphics Corporation.

     THIS FILE CONTAINS PROPRIETARY, CONFIDENTIAL, AND TRADE
     SECRET INFORMATION OF MENTOR GRAPHICS AND/OR ITS LICENSORS.

     You may not use or distribute this file without the express
     written permission of Mentor Graphics or its authorized
     distributor.  This file is licensed only for use with
     Sourcery CodeBench.  No other use is permitted.
     -->

<!ELEMENT board
 (category?, properties?, feature?, initialize?, memory-map?, \
debuggerDefaults?)>

<!-- Board category to group boards list into the tree -->
```

[1] https://sourcery.mentor.com/GNUToolchain/kbentry132

```
<!ELEMENT category (#PCDATA)>

<!ELEMENT properties
 (description?, property*)>

<!ELEMENT initialize
 (write-register | write-memory | delay
  | wait-until-memory-equal | wait-until-memory-not-equal)* >
<!ELEMENT write-register EMPTY>
<!ATTLIST write-register
          address CDATA   #REQUIRED
                        value   CDATA   #REQUIRED
                        bits    CDATA   #IMPLIED>
<!ELEMENT write-memory EMPTY>
<!ATTLIST write-memory
          address CDATA   #REQUIRED
                        value   CDATA   #REQUIRED
                        bits    CDATA   #IMPLIED>
<!ELEMENT delay EMPTY>
<!ATTLIST delay
          time CDATA   #REQUIRED>
<!ELEMENT wait-until-memory-equal EMPTY>
<!ATTLIST wait-until-memory-equal
          address CDATA   #REQUIRED
                        value   CDATA   #REQUIRED
                        timeout CDATA   #IMPLIED
                        bits    CDATA   #IMPLIED>
<!ELEMENT wait-until-memory-not-equal EMPTY>
<!ATTLIST wait-until-memory-not-equal
          address CDATA   #REQUIRED
                        value   CDATA   #REQUIRED
                        timeout CDATA   #IMPLIED
                        bits    CDATA   #IMPLIED>

<!ELEMENT memory-map (memory-device)*>
<!ELEMENT memory-device (property*, description?, sectors*)>
<!ATTLIST memory-device
                        address CDATA   #REQUIRED
          size    CDATA   #REQUIRED
          type    CDATA   #REQUIRED
                        device  CDATA   #IMPLIED>

<!ELEMENT description (#PCDATA)>
<!ELEMENT property (#PCDATA)>
<!ATTLIST property name CDATA #REQUIRED>
<!ELEMENT sectors EMPTY>
<!ATTLIST sectors
 size CDATA #REQUIRED
 count CDATA #REQUIRED>

<!-- Definition of default option values for each debug interface -->
<!ELEMENT debuggerDefaults (debugInterface*)>
<!ELEMENT debugInterface (option*)>
<!ATTLIST debugInterface
```

```
 name CDATA #REQUIRED
>
<!ELEMENT option EMPTY>
<!ATTLIST option
 name CDATA #REQUIRED
 defaultValue CDATA #REQUIRED
>

<!ENTITY % gdbtarget SYSTEM "gdb-target.dtd">
%gdbtarget;
```

All values can be provided in decimal, hex (with a `0x` prefix) or octal (with a `0` prefix). Addresses and memory sizes can use a `K`, `KB`, `M`, `MB`, `G` or `GB` suffix to denote a unit of memory. Times must use a `ms` or `us` suffix.

The following elements are available:

`<board>` — This top-level element encapsulates the entire description of the board. It can contain `<category>`, `<properties>`, `<feature>`, `<initialize>` and `<memory-map>` elements.

`<category>` — The `<category>` element specifies a '.' separated categorization of this board (e.g., Vendor.Family) to allow grouping similar boards in a tree structure.

`<properties>` — The `<properties>` element specifies specific properties of the target system. This element can occur at most once. It can contain a `<description>` element.

`<initialize>` — The `<initialize>` element defines an initialization sequence for the board, which the Sprite performs before downloading a program. It can contain `<write-register>`, `<write-memory>` and `<delay>` elements.

`<feature>` — This element is used to inform GDB about additional registers and peripherals available on the board. It is passed directly to GDB; see the GDB manual for further details.

`<memory-map>` — This element describes the memory map of the target board. It is used by GDB to determine where software breakpoints may be used and when flash programming sequences must be used. This element can occur at most once. It can contain `<memory-device>` elements.

`<memory-device>` — This element specifies a region of memory. It has four attributes: `address`, `size`, `type` and `device`. The `address` and `size` attributes specify the location of the memory device. The `type` attribute specifies that device as `ram`, `rom` or `flash`. The `device` attribute is required for `flash` regions; it specifies the flash device type. The `<memory-device>` element can contain a `<description>` element.

`<write-register>` — This element writes a value to a control register. It has three attributes: `address`, `value` and `bits`. The `bits` attribute, specifying the bit width of the write operation, is optional; it defaults to 32.

`<write-memory>`	This element writes a value to a memory location. It has three attributes: `address`, `value` and `bits`. The `bits` attribute is optional and defaults to 32. Bit widths of 8, 16 and 32 bits are supported. The address written to must be naturally aligned for the size of the write being done.
`<delay>`	This element introduces a delay. It has one attribute, `time`, which specifies the number of milliseconds, or microseconds to delay by.
`<description>`	This element encapsulates a human-readable description of its enclosing element.
`<property>`	The `<property>` element allows additional name/value pairs to be specified. The property name is specified in a `name` attribute. The property value is the body of the `<property>` element.
`<debuggerDefaults>`	The `<debuggerDefaults>` element defines the default option values for each debug interface.

Chapter 6
Next Steps with Sourcery CodeBench

This chapter describes where you can find additional documentation and information about using Sourcery CodeBench Lite and its components.

6.1. Sourcery CodeBench Knowledge Base

The Sourcery CodeBench Knowledge Base is available to registered users at the Sourcery CodeBench Portal[1]. Here you can find solutions to common problems including installing Sourcery CodeBench, making it work with specific targets, and interoperability with third-party libraries. There are also additional example programs and tips for making the most effective use of the toolchain and for solving problems commonly encountered during debugging. The Knowledge Base is updated frequently with additional entries based on inquiries and feedback from customers.

6.2. Manuals for GNU Toolchain Components

Sourcery CodeBench Lite includes the full user manuals for each of the GNU toolchain components, such as the compiler, linker, assembler, and debugger. Most of the manuals include tutorial material for new users as well as serving as a complete reference for command-line options, supported extensions, and the like.

When you install Sourcery CodeBench Lite, links to both the PDF and HTML versions of the manuals are created in the shortcuts folder you select. If you elected not to create shortcuts when installing Sourcery CodeBench Lite, the documentation can be found in the `share/doc/mips-mips-sde-elf/` subdirectory of your installation directory.

In addition to the detailed reference manuals, Sourcery CodeBench Lite includes a Unix-style manual page for each toolchain component. You can view these by invoking the `man` command with the pathname of the file you want to view. For example, you can first go to the directory containing the man pages:

```
> cd $INSTALL/share/doc/mips-mips-sde-elf/man/man1
```

Then you can invoke `man` as:

```
> man ./mips-sde-elf-gcc.1
```

Alternatively, if you use `man` regularly, you'll probably find it more convenient to add the directory containing the Sourcery CodeBench man pages to your `MANPATH` environment variable. This should go in your `.profile` or equivalent shell startup file; see Section 1.7, “Setting up the Environment” for instructions. Then you can invoke `man` with just the command name rather than a pathname.

Finally, note that every command-line utility program included with Sourcery CodeBench Lite can be invoked with a `--help` option. This prints a brief description of the arguments and options to the program and exits without doing further processing.

[1] https://sourcery.mentor.com/GNUToolchain/

Appendix A
Sourcery CodeBench Lite Release Notes

This appendix contains information about changes in this release of Sourcery CodeBench Lite for MIPS ELF. You should read through these notes to learn about new features and bug fixes.

A.1. Changes in Sourcery CodeBench Lite for MIPS ELF

This section documents Sourcery CodeBench Lite changes for each released revision.

A.1.1. Changes in Sourcery CodeBench Lite 2016.05-7

Unexpected termination. An LD bug that caused executables compiled with `-mips16` to terminate unexpectedly has been fixed.

A.1.2. Changes in Sourcery CodeBench Lite 2016.05-3

Fix thread-safe library support. A bug has been fixed that disabled thread safety in the libraries included with Sourcery CodeBench Lite for MIPS ELF.

GCC version 5.3. Sourcery CodeBench Lite for MIPS ELF is now based on GCC version 5.3. For more information about changes from GCC version 5.2 that was included in previous releases, see `https://gcc.gnu.org/gcc-5/changes.html`.

`-mnon-pic-abicalls` support. GAS no longer supports the option `-mnon-pic-abicalls`. `-call_nonpic` can be used instead.

Removal of MIPS ELF converter tool. The MIPS ELF converter tool `mips-sde-elf-conv` has been removed from Sourcery CodeBench Lite.

Fix for silent truncation when out of disk space on Windows. On Microsoft Windows hosts, programs in the GNU binary utilities sometimes failed to report an error when there was insufficient space for the output file, and instead created a truncated file. This is now fixed.

GBD backtrace fix. A bug that caused missing backtrace information in GDB for applications compiled with `-mips16` has been fixed.

GDB wide character printing on Windows. On Microsoft Windows hosts, GDB now prints wide character and wide string values by assuming a UTF-32 encoding. Previously, `wchar_t` strings printed as a sequence of byte values, as if the contents were a non-wide `char` string.

Breakpoint in non-existent file bug fix. A GDB bug that caused an internal error when setting a breakpoint from a Windows host has been fixed. The error was generated when using `break` *`filename`*`:`*`linenumber`* where the *`filename`* string included the Windows logical volume (for example, `C:`), and the file did not exist or was not used in the program.

A.1.3. Changes in Sourcery CodeBench Lite 2015.11-33

GDB backtrace failure. An LD bug that caused GDB backtraces to fail when using the `-mips16` option has been fixed.

DWARF signed LEB128 encoding fix. An assembler bug has been fixed that caused it to generate unnecessarily long encodings for `.sleb128` constants, used in DWARF debug information.

Soft-float link error fix. A CS3 bug has been fixed that caused undefined symbol errors when linking some programs with `-msoft-float` libraries.

GDB separate debug information fix. A GDB bug has been fixed that caused it to fail to locate the separate debug information for executables containing a build ID section.

Installer change error. A bug that affected re-running the installer to update an existing Sourcery CodeBench Lite installation has been fixed. The bug caused selecting `Change` and then `Cancel` to corrupt the existing install.

A.1.4. Changes in Sourcery CodeBench Lite 2015.11-12

GCC version 5.2. Sourcery CodeBench Lite for MIPS ELF is now based on GCC version 5.2. For more information about changes from GCC version 4.9 that was included in previous releases, see `https://gcc.gnu.org/gcc-5/changes.html`. Note that a new release numbering scheme has been adopted beginning with GCC 5; for details see `https://gcc.gnu.org/develop.html#num_scheme`.

Change to GCC `-save-temps` output. GCC no longer forces verbose assembly language output (`-fverbose-asm`) when the `-save-temps` option is used.

Assembler support for XLP. Assembler support for the XLP architecture has been removed.

Binutils update. The binutils package has been updated to version 2.25.51 from the FSF trunk, git revision cdaec3f3e7ea9118204f0e579bd3257234fbae63. This update includes numerous bug fixes.

Removal of IEEE 754-2008 deprecated options. The deprecated options `-mabs2008`, `-mno-abs2008`, `-mnan2008`, and `-mno-nan2008` are no longer supported by GCC and GAS. They have been replaced by the `-mnan=2008>` and the `-mnan=legacy` options. For more information, please refer to the compiler and assembler manuals.

Newlib update. The Newlib package has been updated to version 2.2.0.

GDB internal error fix. A bug has been fixed that caused GDB to crash with an internal error in `get_frame_id` when it encounters invalid data while generating a backtrace. An example of when this can occur is when the current program location is in a hand-coded assembly routine that does not maintain a valid stack frame.

GDB update. The version of GDB has been updated to 7.10.50-cvs, git revision cdaec3f3e7ea9118204f0e579bd3257234fbae63. This update adds numerous bug fixes and features. Refer to `http://www.gnu.org/software/gdb/news` for more information.

Floating-point zero sign fixes. Several bugs causing QEMU to fail to conform to the IEEE 754 standard for floating-point arithmetic were fixed. These bugs caused the results of some arithmetic operations yielding zero to have the incorrect sign.

Failure to find linker fixed. A bug that caused GCC's `-fuse-ld=bfd` option to fail has been fixed.

A.1.5. Changes in Sourcery CodeBench Lite 2015.05-19

Add declaration for `__cs3_reset`. The `__cs3_reset` symbol is now defined in `cs3.h`, allowing applications to perform a soft reset without having to explicitly define that symbol.

GDB simulator removed. The built-in MIPS processor simulator has been removed from the debugger.

Remove XLP Sprite backend. The XLP backend has been removed from the Sourcery CodeBench Debug Sprite.

A.1.6. Changes in Sourcery CodeBench Lite 2015.05-2

GCC version 4.9.2. Sourcery CodeBench Lite for MIPS ELF is now based on GCC version 4.9.2. This update fixes a number of bugs; for details, see `http://gcc.gnu.org/gcc-4.9/changes.html`.

GDB memory corruption bug fix. A bug in support for long pathnames on Windows hosts has been fixed that could lead to memory corruption, causing potential crashes and unpredictable behavior.

GDB source code display fix. A bug has been fixed that prevented GDB from displaying source code from assembly files during single-stepping.

QEMU version 2.2.0. The version of QEMU included with Sourcery CodeBench Lite for MIPS ELF has been updated to version 2.2.0, corresponding to git revision 45e1611de8be0eae55967694dd6e627c2dc354f2. For information about changes in this version, see `http://wiki.qemu.org/ChangeLog/2.2`.

A.1.7. Changes in Sourcery CodeBench Lite 2014.11-24

No significant changes. There are no significant changes for MIPS ELF in this release.

A.1.8. Changes in Sourcery CodeBench Lite 2014.11-21

Bug fix for incorrect code. A bug that caused incorrect code to be generated when compiling with the `-pg` and `-fuse-caller-save` options has been fixed.

Windows reserved filename bug fix. A bug has been fixed that prevented the compiler and other tools from recognizing Microsoft Windows reserved filenames, such as `NUL`.

MIPS new processor support. GCC now supports the MIPS32R3, MIPS32R5, MIPS64R3, and MIPS64R5 processors. You can use the command line options `-march=mips32r3`, `-march=mips32r5`, `-march=mips64r3`, and `-march=mips64r5` to enable code generation for these targets.

MIPS XPA support. GAS now supports the MIPS eXtended Physical Address (XPA) ASE. You can specify the `-mxpa` option to GCC and GAS to indicate that XPA instructions are being used.

CS3 linker script bug fix. A bug in CS3 has been fixed that caused linker scripts for some targets to assign data associated with the `--build-id` linker option to inappropriate memory regions, such as boot ROM.

Optimized `strcmp`. `strcmp` has now been optimized for speed.

Complex variables manipulation fix. A bug has been fixed that caused GDB to corrupt complex variables living in registers whenever their values were modified.

A.1.9. Changes in Sourcery CodeBench Lite 2014.11-10

Long path support on Windows hosts. Sourcery CodeBench Lite on Microsoft Windows hosts now supports source and object files with paths longer than 259 characters. (These were already supported on GNU/Linux hosts.)

Internal compiler error fix. GCC now issues a diagnostic instead of crashing with an internal compiler error on code using `register` and `asm` to assign variables to registers that are reserved

for special purposes (such as the program counter) or are otherwise unsuitable for the type of the variable.

Internal compiler error fix. A bug in GCC has been fixed that sometimes caused internal compiler errors in `var-tracking.c` when compiling with optimization and debug information enabled.

GCC version 4.9.1. Sourcery CodeBench Lite for MIPS ELF is now based on GCC version 4.9.1. For more information about changes from GCC version 4.8 that was included in previous releases, see `http://gcc.gnu.org/gcc-4.9/changes.html`.

Fix for debug information bug. A bug has been fixed that caused errors in the GNU binary utilities, such as running out of memory during linking or displaying incorrect section sizes in objdump. The bug was triggered by objects containing uncompressed debug information for symbols whose names start with `ZLIB`.

Fix for initialization of arguments to `main`. A bug in CS3 has been fixed that caused the values of *`argc`* and *`argv`* passed to `main` to be uninitialized on 64-bit targets.

Newlib update. The Newlib package has been updated to version 2.1.0, with additions from the community CVS trunk as of 2014-06-03.

GDB `set mips compression` command bug fix. A bug in the GDB `set mips compression` command has been fixed that caused it to have no effect.

A.1.10. Changes in Older Releases

For information about changes in older releases of Sourcery CodeBench Lite for MIPS ELF, please refer to the Getting Started guide packaged with those releases.

Appendix B
Sourcery CodeBench Lite Licenses

Sourcery CodeBench Lite contains software provided under a variety of licenses. Some components are “free” or “open source” software, while other components are proprietary. This appendix explains what licenses apply to your use of Sourcery CodeBench Lite. You should read this appendix to understand your legal rights and obligations as a user of Sourcery CodeBench Lite.

The Mentor Graphics License is available in Section B.1, “Sourcery CodeBench Lite License Agreement”.

B.1. Sourcery CodeBench Lite License Agreement

Sourcery CodeBench Lite for MIPS ELF is licensed under the Mentor Graphics **Embedded Software and Hardware License Agreement.** If you have a separate signed or shrinkwrap agreement (as applicable) with Mentor Graphics related to your use of Sourcery CodeBench Lite, your order is subject to the terms of that agreement. If you do not, the following terms apply, unless otherwise specifically agreed to in writing by an authorized representative of Mentor Graphics. The terms of this Getting Started Guide supplement, but do not replace or amend, the terms of your separate agreement with Mentor Graphics. Accordingly, to the extent the following terms and conditions conflict with such separate agreement, the terms and conditions of the separate agreement shall control.

The latest version of the License Agreement is available on-line at http://www.mentor.com/terms_conditions/embedded_software_license.

B.1.1. Embedded Software and Hardware License Agreement

IMPORTANT INFORMATION

USE OF ALL PRODUCTS IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF PRODUCTS INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

EMBEDDED SOFTWARE AND HARDWARE LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Products (as defined in Section 1) between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Products received electronically, certify destruction of Products and all accompanying items within five days after receipt of such Products and receive a full refund of any license fee paid.

1. **Definitions.**

    1.1. "Customer's Product" means Customer's end-user product identified by a unique SKU (including any Related SKUs) in an applicable Addenda that is developed, manufactured, branded and shipped solely by Customer or an authorized manufacturer or subcontractor on behalf of Customer to end-users or consumers;

1.2. "Developer" means a unique user, as identified by a unique user identification number, with access to Embedded Software at an authorized Development Location. A unique user is an individual who works directly with the embedded software in source code form, or creates, modifies or compiles software that ultimately links to the Embedded Software in Object Code form and is embedded into Customer's Product at the point of manufacture;

1.3. "Development Location" means the location where Products may be used as authorized in the applicable Addenda;

1.4. "Development Tools" means the software that may be used by Customer for building, editing, compiling, debugging or prototyping Customer's Product;

1.5. "Embedded Software" means Software that is embeddable;

1.6. "End-User" means Customer's customer;

1.7. "Executable Code" means a compiled program translated into a machine-readable format that can be loaded into memory and run by a certain processor;

1.8. "Hardware" means a physically tangible electro-mechanical system or sub-system and associated documentation;

1.9. "Linkable Object Code" or "Object Code" means linkable code resulting from the translation, processing, or compiling of Source Code by a computer into machine-readable format;

1.10. "Mentor Embedded Linux" or "MEL" means Mentor Graphics' tools, source code, and recipes for building Linux systems;

1.11. "Open Source Software" or "OSS" means software subject to an open source license which requires as a condition for redistribution of such software, including modifications thereto, that the: (i) redistribution be in source code form or be made available in source code form; (ii) redistributed software be licensed to allow the making of derivative works; or (iii) redistribution be at no charge;

1.12. "Processor" means the specific microprocessor to be used with Software and implemented in Customer's Product;

1.13. "Products" means Software, Term-Licensed Products and/or Hardware;

1.14. "Proprietary Components" means the components of the Products that are owned and/or licensed by Mentor Graphics and are not subject to an Open Source Software license, as more fully set forth herein;

1.15. "Redistributable Components" means those components that are intended to be incorporated or linked into Customer's Linkable Object Code developed with the Software, as more fully set forth herein;

1.16. "Related SKU" means two or more Customer Products identified by logically-related SKUs, where there is no difference or change in the electrical hardware or software content between such Customer Products;

1.17. "Software" means software programs, Embedded Software and/or Development Tools, including any updates, modifications, revisions, copies, documentation and design data that are licensed under this Agreement;

1.18. "Source Code" means software in a form in which the program logic is readily understandable by a human being;

1.19. "Sourcery CodeBench Software" means Mentor Graphics' Development Tool for C/C++ embedded application development;

1.20. "Sourcery VSIPL++" is Software providing C++ classes and functions for writing embedded signal processing applications designed to run on one or more processors;

1.21. "Stock Keeping Unit" or "SKU" is a unique number or code used to identify each distinct product, item or service available for purchase;

1.22. "Subsidiary" means any corporation more than 50% owned by Customer, excluding Mentor Graphics competitors. Customer agrees to fulfill the obligations of such Subsidiary in the event of default. To the extent Mentor Graphics authorizes any Subsidiary's use of Products under this Agreement, Customer agrees to ensure such Subsidiary's compliance with the terms of this Agreement and will be liable for any breach by a Subsidiary; and

1.23. "Term-Licensed Products" means Products licensed to Customer for a limited time period ("Term").

2. **Orders, Fees and Payment.**

2.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement and any applicable Addenda, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.

2.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. All invoices will be sent electronically to Customer on the date stated on the invoice unless otherwise specified in an Addendum. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.

2.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

3. **Grant of License.**

3.1. The Products installed, downloaded, or otherwise acquired by Customer under this Agreement constitute or contain copyrighted, trade secret, proprietary and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software as described in the applicable Addenda. The limited licenses granted under the applicable Addenda shall continue until the expiration date of Term-Licensed Products or termination in accordance with Section 12 below, whichever occurs first. Mentor Graphics does NOT grant Customer any right to (a) sublicense or (b) use Software beyond the scope of this Section without first signing a separate agreement or Addenda with Mentor Graphics for such purpose.

3.2. License Type. The license type shall be identified in the applicable Addenda.

3.2.1. Development License: During the Term, if any, Customer may modify, compile, assemble and convert the applicable Embedded Software Source Code into Linkable Object Code and/or Executable Code form by the number of Developers specified, for the Processor(s), Customer's Product(s) and at the Development Location(s) identified in the applicable Addenda.

3.2.2. End-User Product License: During the Term, if any, and unless otherwise specified in the applicable Addenda, Customer may incorporate or embed an Executable Code version of the Embedded Software into the specified number of copies of Customer's Product(s), using the Processor Unit(s), and at the Development Location(s) identified in the applicable Addenda. Customer may manufacture, brand and distribute such Customer's Product(s) worldwide to its End-Users.

3.2.3. Internal Tool License: During the Term, if any, Customer may use the Development Tools solely: (a) for internal business purposes and (b) on the specified number of computer work stations and sites. Development Tools are licensed on a per-seat or floating basis, as specified in the applicable Addenda, and shall not be distributed to others or delivered in Customer's Product(s) unless specifically authorized in an applicable Addenda.

3.2.4. Sourcery CodeBench Professional Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software (i) if the license is a node-locked license, by a single user who uses the Software on up to two machines provided that only one copy of the Software is in use at any one time, or (ii) if the license is a floating license, by the authorized number of concurrent users on one or more machines provided that only the authorized number of copies of the Software are in use at any one time, and (b) distribute the Redistributable Components of the Software in Executable Code form only and only as part of Customer's Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.

3.2.5. Sourcery CodeBench Standard Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software by a single user who uses the Software on up to two machines provided that only one copy of the Software is in use at any one time, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer's Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.

3.2.6. Sourcery CodeBench Personal Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components

of the Software by a single user who uses the Software on one machine, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.

3.2.7. Sourcery CodeBench Academic Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software for non-commercial, academic purposes only by a single user who uses the Software on one machine, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.

3.3. Mentor Graphics may from time to time, in its sole discretion, lend Products to Customer. For each loan, Mentor Graphics will identify in writing the quantity and description of Software loaned, the authorized location and the Term of the loan. Mentor Graphics will grant to Customer a temporary license to use the loaned Software solely for Customer's internal evaluation in a non-production environment. Customer shall return to Mentor Graphics or delete and destroy loaned Software on or before the expiration of the loan Term. Customer will sign a certification of such deletion or destruction if requested by Mentor Graphics.

4. **Beta Code.**

4.1. Portions or all of certain Products may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.

4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **Restrictions on Use.**

5.1. Customer may copy Software only as reasonably necessary to support the authorized use, including archival and backup purposes. Each copy must include all notices and legends

embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Except where embedded in Executable Code form in Customer's Product, Customer shall maintain a record of the number and location of all copies of Software, including copies merged with other software and products, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees, authorized manufacturers or authorized contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics immediate written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use.

5.2. Customer acknowledges that the Products provided hereunder may contain Source Code which is proprietary and its confidentiality is of the highest importance and value to Mentor Graphics. Customer acknowledges that Mentor Graphics may be seriously harmed if such Source Code is disclosed in violation of this Agreement. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any Source Code from Products that are not provided in Source Code form. Except as embedded in Executable Code in Customer's Product and distributed in the ordinary course of business, in no event shall Customer provide Products to Mentor Graphics competitors. Log files, data files, rule files and script files generated by or for the Software (collectively "Files") constitute and/or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Under no circumstances shall Customer use Products or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Products, or disclose to any third party the results of, or information pertaining to, any benchmark.

5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent, which shall not be unreasonably withheld, and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.4. Notwithstanding any provision in an OSS license agreement applicable to a component of the Sourcery CodeBench Software that permits the redistribution of such component to a third party in Source Code or binary form, Customer may not use any Mentor Graphics trademark, whether registered or unregistered, in connection with such distribution, and may not recompile the Open Source Software components with the --with-pkgversion or --with-bugurl configuration options that embed Mentor Graphics' trademarks in the resulting binary.

5.5. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **Support Services.**

   6.1. Except as described in Sections 6.2, 6.3 and 6.4 below, and unless otherwise specified in any applicable Addenda to this Agreement, to the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased in accordance with Mentor Graphics' then-current End-User Software Support Terms located at http://supportnet.mentor.com/about/legal/.

   6.2. To the extent Customer purchases support services for Sourcery CodeBench Software, support will be provided solely in accordance with the provisions of this Section 6.2. Mentor Graphics shall provide updates and technical support to Customer as described herein only on the condition that Customer uses the Executable Code form of the Sourcery CodeBench Software for internal use only and/or distributes the Redistributable Components in Executable Code form only (except as provided in a separate redistribution agreement with Mentor Graphics or as required by the applicable Open Source license). Any other distribution by Customer of the Sourcery CodeBench Software (or any component thereof) in any form, including distribution permitted by the applicable Open Source license, shall automatically terminate any remaining support term. Subject to the foregoing and the payment of support fees, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased in accordance with Mentor Graphics' then-current Sourcery CodeBench Software Support Terms located at http://www.mentor.com/codebench-support-legal.

   6.3. To the extent Customer purchases support services for Sourcery VSIPL++, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased solely in accordance with Mentor Graphics' then-current Sourcery VSIPL++ Support Terms located at. http://www.mentor.com/vsipl-support-legal.

   6.4. To the extent Customer purchases support services for Mentor Embedded Linux, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased solely in accordance with Mentor Graphics' then-current Mentor Embedded Linux Support Terms located at http://www.mentor.com/mel-support-legal.

7. **Third Party and Open Source Software.** Products may contain Open Source Software or code distributed under a proprietary third party license agreement. Please see applicable Products documentation, including but not limited to license notice files, header files or source code for further details. Please see Section B.2.2, “Components” for additional rights and obligations governing your use and distribution of Open Source Software. Customer agrees that it shall not subject any Product provided by Mentor Graphics under this Agreement to any Open Source Software license that does not otherwise apply to such Product. In the event of conflict between the terms of this Agreement, any Addenda and an applicable OSS or proprietary third party agreement, the OSS or proprietary third party agreement will control solely with respect to the OSS or proprietary third party software component. The provisions of this Section 7 shall survive the termination of this Agreement.

8. **Limited Warranty.**

   8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual and/or specification. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after

delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Products under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; OR (B) PRODUCTS PROVIDED AT NO CHARGE, WHICH ARE PROVIDED "AS IS" UNLESS OTHERWISE AGREED IN WRITING.

8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE TO CUSTOMER AND DO NOT APPLY TO ANY END-USER. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, WITH RESPECT TO PRODUCTS OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, AND EXCEPT FOR EITHER PARTY'S BREACH OF ITS CONFIDENTIALITY OBLIGATIONS, CUSTOMER'S BREACH OF LICENSING TERMS OR CUSTOMER'S OBLIGATIONS UNDER SECTION 10, IN NO EVENT SHALL: (A) EITHER PARTY OR ITS RESPECTIVE LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF SUCH PARTY OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES; AND (B) EITHER PARTY OR ITS RESPECTIVE LICENSORS' LIABILITY UNDER THIS AGREEMENT, INCLUDING, FOR THE AVOIDANCE OF DOUBT, LIABILITY FOR ATTORNEYS' FEES OR COSTS, EXCEED THE GREATER OF THE FEES PAID OR OWING TO MENTOR GRAPHICS FOR THE PRODUCT OR SERVICE GIVING RISE TO THE CLAIM OR $500,000 (FIVE HUNDRED THOUSAND U.S. DOLLARS). NOTWITHSTANDING THE FOREGOING, IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **Hazardous Applications.**

10.1. Customer agrees that Mentor Graphics has no control over Customer's testing or the specific applications and use that Customer will make of Products. Mentor Graphics Products are not specifically designed for use in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support systems, medical devices or other applications in which the failure of Mentor Graphics Products could lead to death, personal injury, or severe physical or environmental damage ("Hazardous Applications").

10.2. CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING PRODUCTS USED IN HAZARDOUS APPLICATIONS AND SHALL BE SOLELY

LIABLE FOR ANY DAMAGES RESULTING FROM SUCH USE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF PRODUCTS IN ANY HAZARDOUS APPLICATIONS.

10.3. CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING REASONABLE ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10.1.

10.4. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **Infringement.**

11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay any costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense, and in addition to its obligations under Section 11.1, either (a) replace or modify the Product so that it becomes noninfringing; or (b) procure for Customer the right to continue using the Product. If Mentor Graphics determines that neither of those alternatives is financially practical or otherwise reasonably available, Mentor Graphics may require the return of the Product and refund to Customer any purchase price or license fee(s) paid.

11.3. Mentor Graphics has no liability to Customer if the claim is based upon: (a) the combination of the Product with any product not furnished by Mentor Graphics, where the Product itself is not infringing; (b) the modification of the Product other than by Mentor Graphics or as directed by Mentor Graphics, where the unmodified Product would not infringe; (c) the use of the infringing Product when Mentor Graphics has provided Customer with a current unaltered release of a non-infringing Product of substantially similar functionality in accordance with Subsection 11.2(a); (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells, where the Product itself is not infringing; (f) any Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; (h) Open Source Software, except to the extent that the infringement is directly caused by Mentor Graphics' modifications to such Open Source Software; or (i) infringement by Customer that is deemed willful. In the case of (i), Customer shall reimburse Mentor Graphics for its reasonable attorneys' fees and other costs related to the action.

11.4. THIS SECTION 11 IS SUBJECT TO SECTION 9 ABOVE AND STATES: (A) THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND (B) CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

12. **Termination and Effect of Termination.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized Term.

    12.1. **Termination for Breach.** This Agreement shall remain in effect until terminated in accordance with its terms. Mentor Graphics may terminate this Agreement and/or any licenses granted under this Agreement, and Customer will immediately discontinue use and distribution of Products, if Customer (a) commits any material breach of any provision of this Agreement and fails to cure such breach upon 30-days prior written notice; or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination. For the avoidance of doubt, nothing in this Section 12 shall be construed to prevent Mentor Graphics from seeking immediate injunctive relief in the event of any threatened or actual breach of Customer's obligations hereunder.

    12.2. **Effect of Termination.** Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination or expiration of the Term, Customer will discontinue use and/or distribution of Products, and shall return Hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form, except to the extent an Open Source Software license conflicts with this Section 12.2 and permits Customer's continued use of any Open Source Software portion or component of a Product. Upon termination for Customer's breach, an End-User may continue its use and/or distribution of Customer's Product so long as: (a) the End-User was licensed according to the terms of this Agreement, if applicable to such End-User, and (b) such End-User is not in breach of its agreement, if applicable, nor a party to Customer's breach.

13. **Export.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies. Customer acknowledges that the regulation of product export is in continuous modification by local governments and/or the United States Congress and administrative agencies. Customer agrees to complete all documents and to meet all requirements arising out of such modifications.

14. **U.S. Government License Rights.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

15. **Third Party Beneficiary.** For any Products licensed under this Agreement and provided by Customer to End-Users, Mentor Graphics or the applicable licensor is a third party beneficiary of the agreement between Customer and End-User. Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

16. **Review of License Usage.** Customer will monitor the access to and use of Software. With prior written notice, during Customer's normal business hours, and no more frequently than

once per calendar year, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system, records, accounts and sublicensing documents deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all Customer information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. Such license review shall be at Mentor Graphics' expense unless it reveals a material underpayment of fees of five percent or more, in which case Customer shall reimburse Mentor Graphics for the costs of such license review. Customer shall promptly pay any such fees. If the license review reveals that Customer has made an overpayment, Mentor Graphics has the option to either provide the Customer with a refund or credit the amount overpaid to Customer's next payment. The provisions of this Section 16 shall survive the termination of this Agreement.

17. **Controlling Law, Jurisdiction and Dispute Resolution.** This Agreement shall be governed by and construed under the laws of the State of California, USA, excluding choice of law rules. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the state and federal courts of California, USA. Nothing in this section shall restrict Mentor Graphics' right to bring an action (including for example a motion for injunctive relief) against Customer or its Subsidiary in the jurisdiction where Customer's or its Subsidiary's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

18. **Severability.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

19. **Miscellaneous.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 120305, Part No. 252061

B.2. Licenses and Third-Party Information for Sourcery CodeBench Lite Components

The table below lists the major components of Sourcery CodeBench Lite for MIPS ELF and the license terms which apply to each of these components.

B.2.1. Mentor Graphics Proprietary Components

Components of the Software that are owned and/or licensed by Mentor Graphics and are not subject to a “free software” or “open source” license, such as the GNU Public License. The Mentor Graphics Proprietary Components of the Software include, without limitation, the Sourcery CodeBench Installer, any Sourcery CodeBench Eclipse plug-ins, and any Sourcery CodeBench Debug Sprite.

B.2.2. Components

Some free or open-source components provide documentation or other files under terms different from those shown below. For definitive information about the license that applies to each component, consult the source package corresponding to this release of Sourcery CodeBench Lite. Sourcery CodeBench Lite may contain free or open-source components not included in the list below; for a definitive list, consult the source package corresponding to this release of Sourcery CodeBench Lite.

Component	**License**
GNU Compiler Collection	GNU General Public License 3.0 http://www.gnu.org/licenses/gpl.html
GNU Binary Utilities	GNU General Public License 3.0 http://www.gnu.org/licenses/gpl.html
GNU Debugger	GNU General Public License 3.0 http://www.gnu.org/licenses/gpl.html
Sourcery CodeBench Debug Sprite	Mentor Graphics License
QEMU Emulator	GNU General Public License 2.0 http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
CodeSourcery Common Startup Code Sequence	Mentor Graphics License
Newlib C Library	BSD License https://sourceware.org/newlib/COPYING.NEWLIB
GNU Make	GNU General Public License 2.0 http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
GNU Core Utilities	GNU General Public License 2.0 http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

Important

Although some of the licenses that apply to Sourcery CodeBench Lite are “free software” or “open source software” licenses, none of these licenses impose any obligation on you to reveal the source code of applications you build with Sourcery CodeBench Lite. You can develop proprietary applications and libraries with Sourcery CodeBench Lite.

Sourcery CodeBench Lite may include some third party example programs and libraries in the `share/sourceryg++-mips-sde-elf-examples` subdirectory. These examples are not covered by the Sourcery CodeBench Software License Agreement. To the extent permitted by law, these examples are provided by CodeSourcery as is with no warranty of any kind, including implied warranties of merchantability or fitness for a particular purpose. Your use of each example is governed by the license notice (if any) it contains.

B.2.3. Third-Party Information

Additional attribution and license information for third-party software bundled with Mentor Graphics Proprietary Components can be found in the `share/doc/sourceryg++-mips-sde-elf/pdf/legal/` directory of your Sourcery CodeBench Lite installation.