Sourcery G++ Lite

ColdFire GNU/Linux

Sourcery G++ Lite 4.4-54

Getting Started

Sourcery G++ Lite: ColdFire GNU/Linux: Sourcery G++ Lite 4.4-54: Getting Started

CodeSourcery, Inc.

Abstract

This guide explains how to install and build applications with Sourcery G++ Lite, CodeSourcery's customized, validated, and supported version of the GNU Toolchain. Sourcery G++ Lite includes everything you need for application development, including C and C++ compilers, assemblers, linkers, and libraries.

When you have finished reading this guide, you will know how to use Sourcery G++ from the command line.

Table of Contents

Preface

This preface introduces the Sourcery G++ 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 G++ Lite. This guide provides a step-by-step guide to installing Sourcery G++ Lite and to building simple applications. Parts of this document assume that you have some familiarity with using the command-line interface. If you are an administrator installing Sourcery G++ Lite on a UNIX-like system for all of your users to use, you should also be familiar with the package-management software (such as the Red Hat Package Manager) for your system.

2. Organization

This document is organized into the following chapters and appendices:

Chapter 1, "Quick Start"	This chapter includes a brief checklist to follow when installing and using Sourcery G++ Lite for the first time. You may use this chapter as an abbreviated guide to the rest of this manual.
Chapter 2, "Installation and Configuration"	This chapter describes how to download, install and configure Sourcery G++ Lite. This section describes the available installation options and explains how to set up your environment so that you can build applications.
Chapter 3, "Sourcery G++ Lite for ColdFire GNU/Linux"	This chapter contains information about using Sourcery G++ Lite that is specific to ColdFire GNU/Linux targets. You should read this chapter to learn how to best use Sourcery G++ Lite on your target system.
Chapter 4, "Using Sourcery G++ from the Command Line"	This chapter explains how to build applications with Sourcery G++ 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 5, "Sourcery G++ Debug Sprite"	This chapter describes the use of the Sourcery G++ Debug Sprite for remote debugging. The Sprite is provided for debugging of the Linux or uClinux kernel on the target board. This chapter includes information about the debugging devices and boards supported by the Sprite for ColdFire GNU/Linux.
Chapter 6, "Next Steps with Sourcery G++"	This chapter describes where you can find additional documentation and information about using Sourcery G++ Lite and its components. It also provides information about Sourcery G++ subscriptions. CodeSourcery customers with Sourcery G++ subscriptions receive comprehensive support for Sourcery G++.
Appendix A, "Sourcery G++ Lite Release Notes"	This appendix contains information about changes in this release of Sourcery G++ Lite for ColdFire GNU/Linux. You should read through these notes to learn about new features and bug fixes.
Appendix B, "Sourcery G++ Lite Licenses"	This appendix provides information about the software licenses that apply to Sourcery G++ Lite. Read this appendix

to understand your legal rights and obligations as a user of Sourcery G++ 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
Quick Start

This chapter includes a brief checklist to follow when installing and using Sourcery G++ Lite for the first time. You may use this chapter as an abbreviated guide to the rest of this manual.

Sourcery G++ Lite for ColdFire GNU/Linux is intended for developers working on embedded GNU/Linux applications. It may also be used for Linux kernel development and debugging, or to build a GNU/Linux distribution.

Follow the steps given in this chapter to install Sourcery G++ Lite and build and run your first application program. The checklist given here is not a tutorial and does not include detailed instructions for each step; however, it will help guide you to find the instructions and reference information you need to accomplish each step. Note that this checklist is also oriented towards application debugging rather than kernel debugging.

You can find additional details about the components, libraries, and other features included in this version of Sourcery G++ Lite in Chapter 3, “Sourcery G++ Lite for ColdFire GNU/Linux”.

1.1. Installation and Set-Up

Install Sourcery G++ Lite on your host computer. You may download an installer package from the Sourcery G++ web site[1], or you may have received an installer on CD. The installer is an executable program that pops up a window on your computer and leads you through a series of dialogs to configure your installation. If the installation is successful, it will offer to launch the Getting Started guide. For more information about installing Sourcery G++ Lite, including host system requirements and tips to set up your environment after installation, refer to Chapter 2, “Installation and Configuration”.

1.2. Configuring Sourcery G++ Lite for the Target System

Identify your target libraries. Sourcery G++ Lite supports libraries optimized for different targets. Libraries are selected automatically by the linker, depending on the processor and other options you have specified. Refer to Section 3.2, “Library Configurations” for details.

Install runtime libraries on your target machine. In order to run programs built with the Sourcery G++ runtime libraries on target hardware, you must install these libraries, the Sourcery G++ dynamic linker, and other runtime support files -- collectively referred to as the *sysroot* -- on your GNU/Linux target system. Typically, this involves either using third-party tools to build a complete root filesystem including the Sourcery G++ sysroot, or using special commands when linking or running your program so it can find the sysroot installed in another location on the target. Refer to Section 3.4, “Using Sourcery G++ Lite on GNU/Linux Targets” for full discussion of these options.

1.3. Building Your Program

Build your program with Sourcery G++ command-line tools. Create a simple test program, and follow the directions in Chapter 4, “Using Sourcery G++ from the Command Line” to compile and link it using Sourcery G++ Lite.

1.4. Running and Debugging Your Program

The steps to run or debug your program depend on your target system and how it is configured. Choose the appropriate method for your target.

[1] http://www.codesourcery.com/gnu_toolchains/

Run your program on the ColdFire GNU/Linux target. To run a program using the included Sourcery G++ libraries, you must install the sysroot on the target, as previously discussed. Copy the executable for your program to the target system. The method you use for launching your program depends on how you have installed the libraries and built your program. In some cases, you may need to invoke the Sourcery G++ dynamic linker explicitly. Refer to Section 3.4, “Using Sourcery G++ Lite on GNU/Linux Targets” for details.

Debug your program on the target using GDB server. You can use GDB server on a remote target to debug your program. When debugging a program that uses the included Sourcery G++ libraries, you must use the **gdbserver** executable included in the sysroot, and similar issues with respect to the dynamic linker as discussed previously apply. See Section 3.5, “Using GDB Server for Debugging” for detailed instructions. Once you have started GDB server on the target, you can connect to it from the debugger on your host system. Refer to Section 4.3, “Running Applications from GDB” for instructions on remote debugging from command-line GDB.

Chapter 2
Installation and Configuration

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

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

2.1. Terminology

Throughout this document, the term *host system* refers to the system on which you run Sourcery G++ while the term *target system* refers to the system on which the code produced by Sourcery G++ runs. The target system for this version of Sourcery G++ is `m68k-linux-gnu`.

If you are developing a workstation or server application to run on the same system that you are using to run Sourcery G++, 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.

2.2. System Requirements

2.2.1. Host Operating System Requirements

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

- Microsoft Windows NT 4, Windows 2000, Windows XP, and Windows Vista systems using IA32, AMD64, and EM64T processors.
- GNU/Linux systems using IA32, AMD64, or EM64T processors, including Debian 3.1 (and later), Red Hat Enterprise Linux 3 (and later), and SuSE Enterprise Linux 8 (and later).

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

Installing on Ubuntu and Debian GNU/Linux Hosts

The Sourcery G++ graphical installer is incompatible with the **dash** shell, which is the default `/bin/sh` for recent releases of the Ubuntu and Debian GNU/Linux distributions. To install Sourcery G++ Lite on these systems, you must make `/bin/sh` a symbolic link to one of the supported shells: **bash**, **csh**, **tcsh**, **zsh**, or **ksh**.

For example, on Ubuntu systems, the recommended way to do this is:

```
> sudo dpkg-reconfigure -plow dash
Install as /bin/sh? No
```

This is a limitation of the installer and uninstaller only, not of the installed Sourcery G++ Lite toolchain.

2.2.2. Host Hardware Requirements

In order to install and use Sourcery G++ Lite, you must have at least 128MB of available memory.

The amount of disk space required for a complete Sourcery G++ Lite installation directory depends on the host operating system and the number of target libraries included. Typically, you should plan on at least 400MB.

In addition, the graphical installer requires a similar amount of temporary 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 `IATEMPDIR` environment variable, or `/tmp` if that is not set.

2.2.3. Target System Requirements

See Chapter 3, “Sourcery G++ Lite for ColdFire GNU/Linux” for requirements that apply to the target system.

2.3. Downloading an Installer

If you have received Sourcery G++ Lite on a CD, or other physical media, then you do not need to download an installer. You may skip ahead to Section 2.4, “Installing Sourcery G++ Lite”.

You can download Sourcery G++ Lite from the Sourcery G++ web site[1]. This free version of Sourcery G++, 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 G++ Portal[2]. For more information about subscriptions for Sourcery G++ product releases, see Section 6.1, “Sourcery G++ Subscriptions”.

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 G++ installer is provided as an executable with the `.exe` extension. For GNU/Linux systems Sourcery G++ Lite is provided as an executable installer package with the `.bin` extension. If installing on a RPM-based GNU/Linux system you may download the `.rpm` file. You may also install from a compressed archive with the `.tar.bz2` extension.

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

2.4. Installing Sourcery G++ Lite

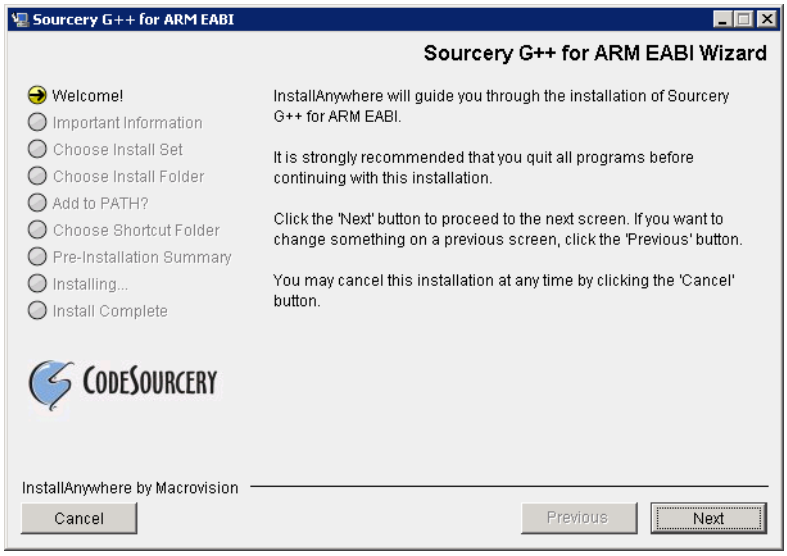
The method used to install Sourcery G++ Lite depends on your host system and the kind of installation package you have downloaded.

2.4.1. Using the Sourcery G++ Lite Installer on Microsoft Windows

If you have received Sourcery G++ Lite on CD, insert the CD in your computer. On most computers, the installer then starts automatically. If your computer has been configured not to automatically run CDs, open `My Computer`, and double click on the CD. If you downloaded Sourcery G++ Lite, double-click on the installer.

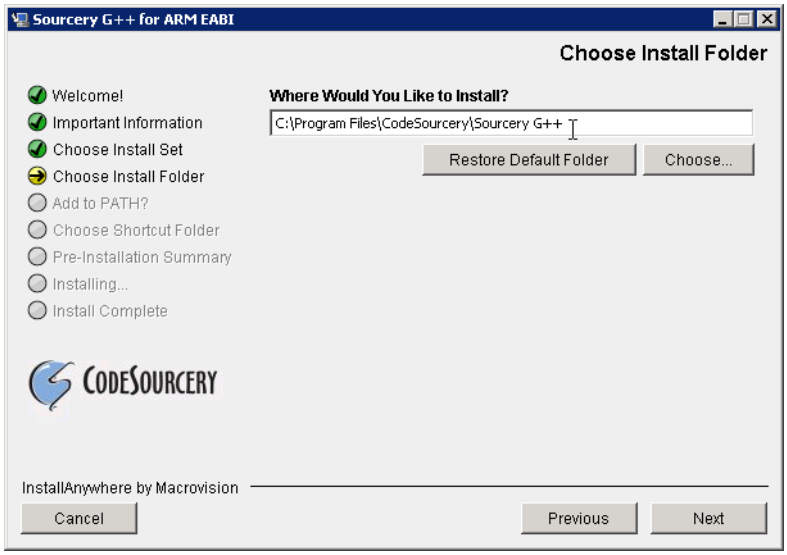
After the installer starts, follow the on-screen dialogs to install Sourcery G++ Lite. The installer is intended to be self-explanatory and on most pages the defaults are appropriate.

[1] http://www.codesourcery.com/gnu_toolchains/
[2] https://support.codesourcery.com/GNUToolchain/

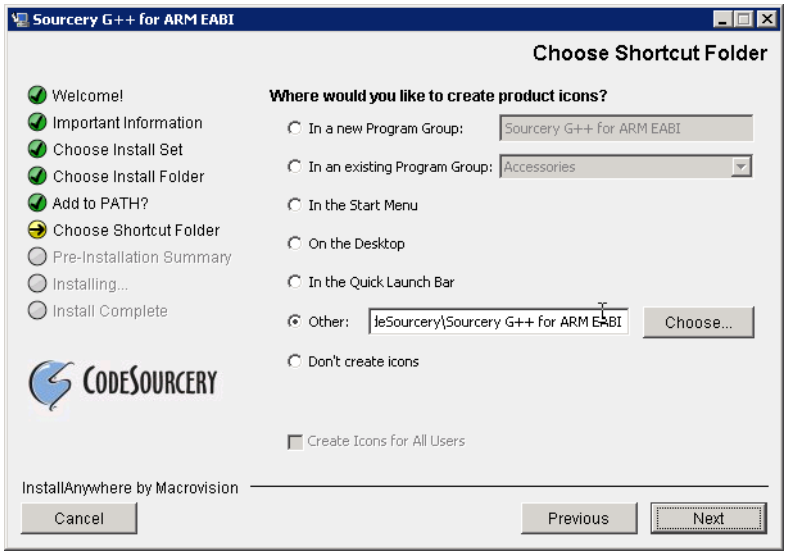


Running the Installer. The graphical installer guides you through the steps to install Sourcery G++ Lite.

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

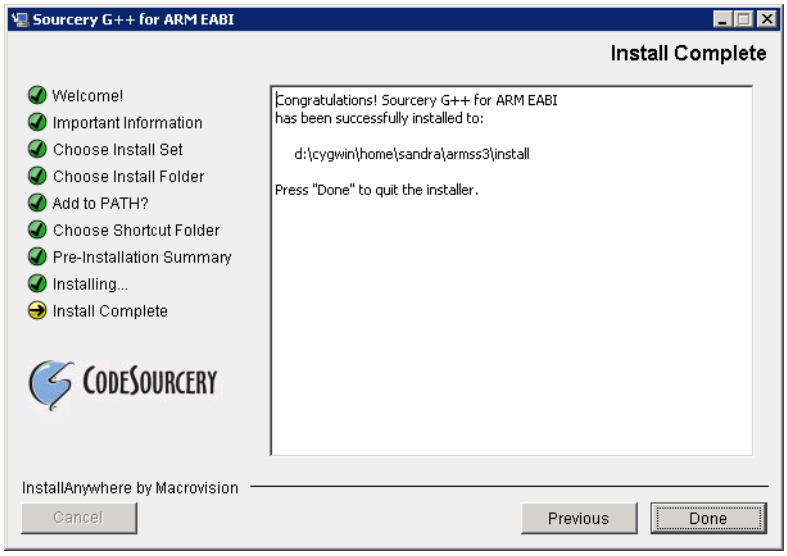


Choose Install Folder. Select the pathname to your install directory.



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

When the installer has finished, it asks if you want to launch a viewer for the Getting Started guide. Finally, the installer displays a summary screen to confirm a successful install before it exits.



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 `-i console` command-line option. For example:

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

2.4.2. Using the Sourcery G++ Lite Installer on GNU/Linux Hosts

Start the graphical installer by invoking the executable shell script:

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

After the installer starts, follow the on-screen dialogs to install Sourcery G++ 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 `-i console` command-line option. For example:

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

2.4.3. Installing Sourcery G++ Lite on RPM-based GNU/Linux Systems

On a RPM-based system you should use RPM to install the provided package. Execute the following command as root (administrator):

```
> rpm -ivh /path/to/package.rpm
```

To update an existing Sourcery G++ Lite installation, use:

```
> rpm -Uvh /path/to/package.rpm
```

2.4.4. Installing Sourcery G++ Lite from a Compressed Archive

You do not need to be a system administrator to install Sourcery G++ Lite from a compressed archive. You may install Sourcery G++ 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 G++ Lite in the `$HOME/CodeSourcery` subdirectory of your home directory and that the filename of the package you have downloaded is `/path/to/package.tar.bz2`. After installation the toolchain will be in `$HOME/CodeSourcery/sourceryg++-4.4`.

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/CodeSourcery
```

Change to the installation directory:

```
> cd $HOME/CodeSourcery
```

Unpack the package:

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

2.5. Installing Sourcery G++ Lite Updates

If you have already installed an earlier version of Sourcery G++ Lite for ColdFire GNU/Linux on your system, it is not necessary to uninstall it before using the installer to unpack a new version in the same location. The installer detects that it is performing an update in that case.

To update a previous RPM installation of Sourcery G++ Lite, use **rpm -U** instead of **rpm -i**, as described above.

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 G++ commands for the ColdFire GNU/Linux target all begin with **m68k-linux-gnu**. This means that you can install Sourcery G++ for multiple target systems in the same directory without conflicts.

2.6. Setting up the Environment

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

2.6.1. Setting up the Environment on Microsoft Windows Hosts

2.6.1.1. Setting the PATH

In order to use the Sourcery G++ tools from the command line, you should add them to your PATH. You may skip this step if you used the graphical installer, since the installer automatically adds Sourcery G++ to your PATH.

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

```
> setx PATH "%PATH%;C:\Program Files\Sourcery G++\bin"
```

where C:\Program Files\Sourcery G++ should be changed to the path of your Sourcery G++ Lite installation.

To set the PATH on a system running a Microsoft Windows version other than Vista, from the desktop bring up the Start menu and right click on My Computer. Select Properties, go to the Advanced tab, then click on the Environment Variables button. Select the PATH variable and click the Edit. Add the string ;C:\Program Files\Sourcery G++\bin to the end, and click OK. Again, you must adjust the pathname to reflect your installation directory.

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

```
> m68k-linux-gnu-g++ -v
```

Verify that the last line of the output contains: Sourcery G++ Lite 4.4-54.

2.6.1.2. Working with Cygwin

Sourcery G++ Lite does not require Cygwin or any other UNIX emulation environment. You can use Sourcery G++ directly from the Windows command shell. You can also use Sourcery G++ 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 G++ is not a Cygwin application, it does not, by default, recognize Cygwin paths.

If you are using Sourcery G++ from Cygwin, you should set the CYGPATH environment variable. If this environment variable is set, Sourcery G++ 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 G++ relies on the **cygpath** utility provided with Cygwin. You must provide Sourcery G++ with the full path to `cygpath` if **cygpath** is not in your `PATH`. For example:

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

directs Sourcery G++ 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.

2.6.2. Setting up the Environment on GNU/Linux Hosts

If you installed Sourcery G++ Lite using the graphical installer then you may skip this step. The installer does this setup for you.

Before using Sourcery G++ 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 $HOME/CodeSourcery/Sourcery_G++/bin:$PATH
```

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

```
> PATH=$HOME/CodeSourcery/Sourcery_G++/bin:$PATH
> export PATH
```

If you are not sure which shell you are using, try both commands. In both cases, if you have installed Sourcery G++ Lite in an alternate location, you must replace the directory above with `bin` subdirectory of the directory in which you installed Sourcery G++ Lite.

You may also wish to set the `MANPATH` environment variable so that you can access the Sourcery G++ manual pages, which provide additional information about using Sourcery G++. To set the `MANPATH` environment variable, follow the same steps shown above, replacing `PATH` with `MANPATH`, and `bin` with `share/doc/sourceryg++-m68k-linux-gnu/man`.

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

```
> m68k-linux-gnu-g++ -v
```

Verify that the last line of the output contains: `Sourcery G++ Lite 4.4-54`.

2.7. Uninstalling Sourcery G++ Lite

The method used to uninstall Sourcery G++ 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 may remove the files you have altered.

2.7.1. Using the Sourcery G++ Lite Uninstaller on Microsoft Windows

For Windows hosts other than Microsoft Windows Vista, select `Start`, then `Control Panel`. Select `Add or Remove Programs`. Scroll down and click on `Sourcery G++ for ColdFire GNU/Linux`. Select `Change/Remove` and follow the on-screen dialogs to uninstall Sourcery G++ Lite.

On Microsoft Windows Vista hosts, select `Start`, then `Settings` and finally `Control Panel`. Select the `Uninstall a program` task. Scroll down and double click on `Sourcery G++ for ColdFire GNU/Linux`. Follow the on-screen dialogs to uninstall Sourcery G++ Lite.

You can run the uninstaller in console mode, rather than using the graphical interface, by invoking the Uninstall executable found in your Sourcery G++ Lite installation directory with the `-i console` command-line option.

To uninstall third-party drivers bundled with Sourcery G++ Lite, first disconnect the associated hardware device. Then use `Add or Remove Programs` (non-Vista) or `Uninstall a program` (Vista) to remove the drivers separately. Depending on the device, you may need to reboot your computer to complete the driver uninstall.

2.7.2. Using the Sourcery G++ Lite Uninstaller on GNU/Linux

You should use the provided uninstaller to remove a Sourcery G++ Lite installation originally created by the executable installer script. The `m68k-linux-gnu` directory located in the install directory will be removed entirely by the uninstaller. Please back up any changes you have made to this directory, such as modified linker scripts.

Start the graphical uninstaller by invoking the executable Uninstall shell script located in your installation directory. After the uninstaller starts, follow the on-screen dialogs to uninstall Sourcery G++ Lite.

You can run the uninstaller in console mode, rather than using the graphical interface, by invoking the Uninstall script with the `-i console` command-line option.

2.7.3. Uninstalling Sourcery G++ Lite on RPM-based GNU/Linux Systems

If you installed Sourcery G++ Lite from an RPM package, you should also use RPM to uninstall it. Execute the following command as root (administrator):

```
> rpm -e sourceryg++-m68k-linux-gnu
```

2.7.4. Uninstalling a Compressed Archive Installation

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

Chapter 3
Sourcery G++ Lite for ColdFire GNU/Linux

This chapter contains information about features of Sourcery G++ Lite that are specific to ColdFire GNU/Linux targets. You should read this chapter to learn how to best use Sourcery G++ Lite on your target system.

3.1. Included Components and Features

This section briefly lists the important components and features included in Sourcery G++ Lite for ColdFire GNU/Linux, and tells you where you may find further information about these features.

Component	Version	Notes
GNU programming tools		
GNU Compiler Collection	4.4.1	Separate manual included.
GNU Binary Utilities	2.19.51	Includes assembler, linker, and other utilities. Separate manuals included.
Debugging support and simulators		
GNU Debugger	6.8.50	Separate manual included.
Sourcery G++ Debug Sprite for ColdFire	4.4-54	Provided for kernel debugging only. See Chapter 5, “Sourcery G++ Debug Sprite”.
CCS Server	N/A	Included with the Sourcery G++ Debug Sprite. See Section 5.5, “Command Converter Server Devices”.
GDB Server	N/A	Included with GDB. See Section 3.5, “Using GDB Server for Debugging”.
Target libraries		
GNU C Library	2.10	Separate manual included.
Linux Kernel Headers	2.6.30	
OpenMP	N/A	
Other utilities		
GNU Make	N/A	Build support on Windows hosts.
GNU Core Utilities	N/A	Build support on Windows hosts.

3.2. Library Configurations

Sourcery G++ 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 G++ selects the multilib matching the build options you have selected.

Each multilib corresponds to a *sysroot* directory that contains the files that should be installed on the target system. The sysroot contains the dynamic linker used to run your applications on the target as well as the libraries. Refer to Section 3.4, “Using Sourcery G++ Lite on GNU/Linux Targets” for instructions on how to install and use these support files on your target GNU/Linux system. You can find the sysroot directories provided with Sourcery G++ in the `m68k-linux-gnu/libc` directory of your installation. In the tables below, the dynamic linker pathname is given relative to the corresponding sysroot.

3.2.1. Included Libraries

The following library configurations are available in Sourcery G++ Lite for ColdFire GNU/Linux.

MCF547X/8X	
Command-line option(s):	default
Sysroot subdirectory:	`./`
Dynamic linker:	`lib/ld.so.1`
Notes:	This multilib supports Freescale MCF547X and MCF548X cores.

MCF54455	
Command-line option(s):	`-mcpu=54455`
Sysroot subdirectory:	`m54455/`
Dynamic linker:	`lib/ld.so.1`
Notes:	This multilib supports Freescale MCF5441x and MCF5445x cores.

3.2.2. Library Selection

A given multilib may be compatible with additional processors and build options beyond those listed above. However, even if a particular set of command-line options produces code compatible with one of the provided multilibs, those options may not be sufficient to identify the intended library to the linker. For example, on some targets, specifying only a processor option on the command line may imply architecture features or floating-point support for compilation, but not for library selection. The details of the mapping from command-line options to multilibs are target-specific and quite complex. Therefore, it is recommended that your link command line include exactly the options listed in the tables above for your intended target multilib. In some cases, you may need to supply different options for linking than for compilation.

If you are uncertain which multilib is selected by a particular set of command-line options, GCC can tell you if you invoke it with the `-print-multi-directory` option in addition to your other build options. For example:

```
> m68k-linux-gnu-gcc -print-multi-directory options...
```

The output of this command is a directory name for the multilib, which you can look up in the tables given previously.

3.3. Target Kernel Requirements

The primary source for Linux kernels for ColdFire targets is the set of LTIB distributions provided on the Freescale web site[1]; the ColdFire changes have not yet been incorporated into the upstream Linux kernel sources. At the time this release was prepared, the kernel in the current LTIB distribution for MCF5445x was based on 2.6.23 kernel, while that for MCF547x/8x was based on 2.6.25. These are the minimal versions for the respective platforms.

This release of Sourcery G++ Lite for ColdFire GNU/Linux includes NPTL (Native POSIX Thread Library) support in the GNU C Library. Supporting NPTL requires that you apply an additional patch to the kernel sources and rebuild the kernel. This release of Sourcery G++ is not compatible with older or unpatched kernels, even if your applications do not use NPTL features.

You can find the required kernel patch and instructions for applying it in the Sourcery G++ Knowledge Base[2].

[1] http://www.freescale.com/
[2] https://support.codesourcery.com/GNUToolchain/kbentry52

3.4. Using Sourcery G++ Lite on GNU/Linux Targets

In order to run and debug programs produced by Sourcery G++ on a GNU/Linux target, you must install runtime support files on the target. You may also need to set appropriate build options so that your executables can find the correct dynamic linker and libraries at runtime.

The runtime support files, referred to as the *sysroot*, are found in the `m68k-linux-gnu/libc` directory of your Sourcery G++ Lite installation. The sysroot consists of the contents of the `etc`, `lib`, `sbin`, and `usr` directories. There may be other directories in `m68k-linux-gnu/libc` that contain additional sysroots customized for particular combinations of command-line compiler flags, or *multilibs*. Refer to Section 3.2, “Library Configurations” for a list of the included multilibs in this version of Sourcery G++ Lite, and the corresponding sysroot directory pathnames.

Note for Windows Host Users

The sysroots provided in Windows host packages for Sourcery G++ are not directly usable on the Linux target because of differences between the Windows and Linux file systems. Some files that are hard links, or copies, in the sysroot as installed on the Windows file system should be symbolic links on the Linux target. Additionally, some files in the sysroot which should be marked executable on the Linux target are not marked executable on Windows. If you intend to use the sysroot provided with Sourcery G++ on a Windows host system as the basis for your Linux target filesystem, you must correct these issues after copying the sysroot to the target. If you are a Professional Edition customer and need assistance with these modifications, please contact CodeSourcery's support team.

There are three choices for installing the sysroot on the target:

- You can install the files in the filesystem root on the target (that is, installing the files directly in `/etc/`, `/lib/`, and so on). All applications on the target then automatically use the Sourcery G++ libraries. This method is primarily useful when you are building a GNU/Linux root filesystem from scratch. If your target board already has a GNU/Linux filesystem installed, overwriting the existing C library files is not recommended, as this may break other applications on your system, or cause it to fail to boot.

- You can install the sysroot in an alternate location and build your application with the `-rpath` and `--dynamic-linker` linker options to specify the sysroot location.

- You can install the sysroot in an alternate location and explicitly invoke your application through the dynamic linker to specify the sysroot location. If you are just getting started with Sourcery G++ Lite, this may be the easiest way to get your application running, but this method does not support use of the debugger.

Setting the environment variable `LD_LIBRARY_PATH` on the target is not sufficient, since executables produced by Sourcery G++ depend on the Sourcery G++ dynamic linker included in the sysroot as well as the Sourcery G++ runtime libraries.

3.4.1. Installing the Sysroot

If you are modifying an existing system, rather than creating a new system from scratch, you should place the sysroot files in a new directory, rather than in the root directory of your target system.

If you choose to overwrite your existing C library, you may not be able to boot your system. You should back up your existing system before overwriting the C library and ensure that you can restore the backup even with your system offline.

When running Sourcery G++ on a GNU/Linux host, you have the alternative of installing the sysroot on the target at the same pathname where it is installed on the host system. One way to accomplish this is to NFS-mount the installation directory on both machines in the same location, rather than to copy files.

In many cases, you do not need to copy all of the files in the sysroot. For example, the `usr/include` subdirectory contains files that are only needed if you will actually be running the compiler on your target system. You do not need these files for non-native compilers. You also do not need any `.o` or `.a` files; these are used by the compiler when linking programs, but are not needed to run programs. You should definitely copy all `.so` files and the executable files in `usr/bin` and `sbin`.

You need to install the sysroot(s) corresponding to the compiler options you are using for your applications. The tables in Section 3.2, "Library Configurations" tell you which sysroot directories correspond to which compiler options. If you are unsure what sysroot is being referenced when you build your program, you can identify the sysroot by adding `-v` to your compiler command-line options, and looking at the `--sysroot=` pathname in the compiler output.

3.4.2. Using Linker Options to Specify the Sysroot Location

If you have installed the sysroot on the target in a location other than the file system root, you can use the `-rpath` and `--dynamic-linker` linker options to specify the sysroot location.

If you are using Sourcery G++ from the command line, follow these steps:

1. First find the correct sysroot directory, dynamic linker, and library subdirectory for your selected multilib. Refer to Section 3.2, "Library Configurations". In the following steps, *`sysroot`* is the absolute path to the sysroot directory on the target corresponding to your selected multilib. For the default multilib, the dynamic linker path relative to the sysroot is `lib/ld.so.1`, and the library subdirectory is `lib`. This is used in the example below.

2. When invoking **m68k-linux-gnu-gcc** to link your executable, include the command-line options:

   ```
   -Wl,-rpath=sysroot/lib:sysroot/usr/lib \
   -Wl,--dynamic-linker=sysroot/lib/ld.so.1
   ```

   where *`sysroot`* is the absolute path to the sysroot directory on the target corresponding to your selected multilib.

3. Copy the executable to the target and execute it normally.

Note that if you specify an incorrect path for `--dynamic-linker`, the common failure mode seen when running your application on the target is similar to

```
> ./factorial
./factorial: No such file or directory
```

or

```
> ./factorial
./factorial: bad ELF interpreter: No such file or directory
```

This can be quite confusing since it appears from the error message as if it is the `./factorial` executable that is missing rather than the dynamic linker it references.

3.4.3. Specifying the Sysroot Location at Runtime

You can invoke the Sourcery G++ dynamic linker on the target to run your application without having to compile it with specific linker options.

To do this, follow these steps:

1. Build your application on the host, without any additional linker options, and copy the executable to your target system.

2. Find the correct sysroot directory, dynamic linker, and library subdirectory for your selected multilib. Refer to Section 3.2, “Library Configurations”. In the following steps, *`sysroot`* is the absolute path to the sysroot directory on the target corresponding to your selected multilib. For the default multilib, the dynamic linker is `lib/ld.so.1`, and the library subdirectory is `lib`. This is used in the example below.

3. On the target system, invoke the dynamic linker with your executable as:

   ```
   > sysroot/lib/ld.so.1 \
     --library-path sysroot/lib:sysroot/usr/lib \
     /path/to/your-executable
   ```

   where *`sysroot`* is the absolute path to the sysroot directory on the target corresponding to your selected multilib.

   Invoking the linker in this manner requires that you provide either an absolute pathname to your executable, or a relative pathname prefixed with `./`. Specifying only the name of a file in the current directory does not work.

3.5. Using GDB Server for Debugging

The GDB server utility provided with Sourcery G++ Lite can be used to debug a GNU/Linux application. While Sourcery G++ runs on your host system, **gdbserver** and the target application run on your target system. Even though Sourcery G++ and your application run on different systems, the debugging experience when using **gdbserver** is very similar to debugging a native application.

3.5.1. Running GDB Server

The GDB server executables are included in the sysroot in ABI-specific subdirectories of *`sysroot`*`/usr`. Use the executable from the sysroot and library subdirectory that match your program. See Section 3.2, “Library Configurations” for details.

You must copy the sysroot to your target system as described in Section 3.4.1, “Installing the Sysroot”. You must also copy the executable you want to debug to your target system.

If you have installed the sysroot in the root directory of the filesystem on the target, you can invoke **gdbserver** as:

```
> gdbserver :10000 program arg1 arg2 ...
```

where *program* is the path to the program you want to debug and *arg1 arg2 ...* are the arguments you want to pass to it. The `:10000` argument indicates that **gdbserver** should listen for connections from GDB on port 10000. You can use a different port, if you prefer.

If you have installed the sysroot in an alternate directory, invoking **gdbserver** becomes more complicated. You must build your application using the link-time options to specify the location of the sysroot, as described in Section 3.4.2, "Using Linker Options to Specify the Sysroot Location". You must also invoke **gdbserver** itself using the dynamic linker provided in the Sourcery G++ sysroot, as described in Section 3.4.3, "Specifying the Sysroot Location at Runtime". In other words, the command to invoke **gdbserver** in this case would be similar to:

```
> sysroot/lib/ld.so.1 \
  --library-path sysroot/lib:sysroot/usr/lib \
  sysroot/usr/lib/bin/gdbserver :10000 program arg1 arg2 ...
```

3.5.2. Connecting to GDB Server from the Debugger

You can connect to GDB server by using the following command from within GDB:

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

where *target* is the host name or IP address of your target system.

When your program exits, **gdbserver** exits too. If you want to debug the program again, you must restart **gdbserver** on the target. Then, in GDB, reissue the `target` command shown above.

3.5.3. Setting the Sysroot in the Debugger

In order to debug shared libraries, GDB needs to map the pathnames of shared libraries on the target to the pathnames of equivalent files on the host system. Debugging of multi-threaded applications also depends on correctly locating copies of the libraries provided in the sysroot on the host system.

In some situations, the target pathnames are valid on the host system. Otherwise, you must tell GDB how to map target pathnames onto the equivalent host pathnames.

In the general case, there are two GDB commands required to set up the mapping:

```
(gdb) set sysroot-on-target target-pathname
(gdb) set sysroot host-pathname
```

This causes GDB to replace all instances of the *target-pathname* prefix in shared library pathnames reported by the target with *host-pathname* to get the location of the equivalent library on the host.

If you have installed the sysroot in the root filesystem on the target, you can omit the **set sysroot-on-target** command, and use only **set sysroot** to specify the location on the host system.

Refer to Section 3.4.1, "Installing the Sysroot" for more information about installing the sysroot on the target. Note that if you have installed a stripped copy of the provided libraries on the target, you should give GDB the location of an unstripped copy on the host.

3.6. Using OpenMP

Sourcery G++ Lite for ColdFire GNU/Linux includes the GNU OpenMP library (libgomp). This is an API that supports multi-platform shared-memory parallel programming.

To compile programs that use OpenMP features, use the `-fopenmp` command-line option. For more information about OpenMP, see `http://www.openmp.org/`.

Chapter 4
Using Sourcery G++ from the Command Line

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

4.1. Building an Application

This chapter explains how to build an application with Sourcery G++ Lite using the command line. As elsewhere in this manual, this section assumes that your target system is m68k-linux-gnu, as indicated by the **m68k-linux-gnu** 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:

```
> m68k-linux-gnu-gcc -o factorial main.c
```

There should be no output from the compiler. (If you are building a C++ application, instead of a C application, replace **m68k-linux-gnu-gcc** with **m68k-linux-gnu-g++**.)

4.2. Running Applications on the Target System

You may need to install the Sourcery G++ runtime libraries and dynamic linker on the target system before you can run your application. Refer to Chapter 3, “Sourcery G++ Lite for ColdFire GNU/Linux” for specific instructions.

To run your program on a GNU/Linux target system, use the command:

```
> factorial
```

You should see:

```
factorial(0) = 1
factorial(1) = 1
factorial(2) = 2
factorial(3) = 6
factorial(4) = 24
factorial(5) = 120
factorial(6) = 720
factorial(7) = 5040
```

```
factorial(8) = 40320
factorial(9) = 362880
```

4.3. 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.

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.

4.3.1. Connecting to the Sourcery G++ Debug Sprite

The Sourcery G++ 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 G++ Debug Sprite” for detailed information about the supported devices.

You can start the Sprite directly from within GDB:

```
(gdb) target remote | m68k-linux-gnu-sprite arguments
```

Refer to Section 5.2, “Invoking Sourcery G++ Debug Sprite” for a full description of the Sprite arguments.

4.3.2. Connecting to an External GDB Server

Sourcery G++ Lite includes a program called **gdbserver** that can be used to debug a program running on a remote ColdFire GNU/Linux target. Follow the instructions in Chapter 3, “Sourcery G++ Lite for ColdFire GNU/Linux” to install and run **gdbserver** on your target system.

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.

Chapter 5
Sourcery G++ Debug Sprite

This chapter describes the use of the Sourcery G++ Debug Sprite for remote debugging. The Sprite is provided for debugging of the Linux or uClinux kernel on the target board. This chapter includes information about the debugging devices and boards supported by the Sprite for ColdFire GNU/Linux.

Sourcery G++ Lite contains the Sourcery G++ Debug Sprite for ColdFire GNU/Linux. 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 5.2, “Invoking Sourcery G++ Debug Sprite” for information about the specific devices supported by this version of Sourcery G++ Lite.

Note for Linux/uClinux users

The Debug Sprite provided with Sourcery G++ Lite allows remote debugging of the Linux or uClinux kernel running on the target. For remote debugging of application programs, you should use **gdbserver** instead. See Chapter 3, “Sourcery G++ Lite for ColdFire GNU/Linux” for details about how to install and run **gdbserver** on the target.

Important

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

5.1. Probing for Debug Devices

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

```
> m68k-linux-gnu-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:

```
CodeSourcery ColdFire Debug Sprite
    (Sourcery G++ Lite Sourcery G++ Lite 4.4-54)
pe: [speed=<n:0-31>&memory-timeout=<n:0-99>] P&E Adaptor
  pe://USBMultilink/PE6011970 - USB1 : USB-ML-CF Rev C (PE6011970)
  pe://CycloneProMaxEthernet/10.0.0.85 - 10.0.0.85  :  cyclone1
ccs: [timeout=<n>&speed=<n>] CCS Protocol
  ccs://$Host:$Port/$Chain_position - CCS address
tblcf: TBLCF Interface
  tblcf://:0/ - TBLCF device
osbdm: Open Source BDM
  osbdm://0/ - OSBDM device
```

This shows that P&E, Command Converter Server (CCS), Turbo BDM Light ColdFire (TBLCF), and Open Source BDM (OSBDM) devices are supported. Two P&E devices are detected, one TBLCF device, and one OSBDM device. Although CCS devices are supported, they cannot be autodetected.

Note that it may take several seconds for the Debug Sprite to probe for all types of supported devices.

5.2. Invoking Sourcery G++ Debug Sprite

The Debug Sprite is invoked as follows:

```
> m68k-linux-gnu-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 G++ Lite for ColdFire GNU/Linux:

`pe` Use a P&E Microcomputer Systems debugging device. Refer to Section 5.4, “P&E Devices”.

`ccs` Use a debugging device controlled by the Command Converter Server (CCS) utility, such as a CodeWarrior Ethernet TAP or USB TAP. Refer to Section 5.5, “Command Converter Server Devices”.

`tblcf` Use a Turbo BDM Light ColdFire (e.g. Axiom AxBDM) debugging device. Refer to Section 5.6, “Turbo BDM Light ColdFire Devices”.

`osbdm` Use an Open Source BDM debugging device. Refer to Section 5.7, “Open Source BDM 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 5.10, “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 5.11, “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.

5.3. Sourcery G++ Debug Sprite Options

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

`-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 | m68k-linux-gnu-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.

5.4. P&E Devices

P&E debug devices are supported. The P&E device partitions the *`device-url`* as follows:

```
pe:[//type[:number]][/key][?device-options]
```

The various parts are:

`type` Specify the debug device type. The following debug device types are supported

- `USBMultilink`
- `CycloneProMaxUSB`
- `CycloneProMaxSerial`
- `CycloneProMaxEthernet`
- `ParallelPortCable`
- `PCIBDMLightning`

`number` Specify the debug device number. Be aware that a device's number depends on whether other devices are concurrently accessed (this is a feature of the underlying P&E library).

`key` Some P&E devices report unique device keys. This option allows you to select a device by its key, independently of USB device numbering.

Not all the separate parts of the *`device-url`* are required to uniquely define a particular device. If you specify more than required, the URL must be self-consistent. If you specify fewer components than required, the Sprite uses the first P&E device found that satisfies the specified components.

The *key* is the most robust mechanism for specifying a device, as it uses the unique ID of a particular P&E device. It is immune from renumbering issues, should boards be unplugged or inserted.

The following *device-options* are permitted:

`speed=`*speed* — Specify the speed of the connection. This is a clock divider value, so higher values are slower connection speeds. Refer to the P&E documentation for valid speed settings for your board.

`memory-timeout=`*timeout* — Some boards report memory errors for every access within a certain time of a genuine memory error. This option instructs the Sprite to compensate for this and retry a memory access that reports an error within the specified time of a prior error. If you need to use this option you need to increase GDB's protocol timeout by specifying `set remotetimeout` *N* at the GDB prompt.

`debug=`*file* — Write P&E debug information to *file*.

5.4.1. Connection Problems

If you get a message “Cannot load P&E library 'UNIT_CFZ.DLL'” or “Cannot load P&E library 'libUnit_cfz.so'”, you probably have not installed the P&E device software. This software is included with Sourcery G++ Lite; see Section 5.4.2, “Installing P&E Drivers” for installation instructions.

The message “Cannot find a matching debug device” means that no P&E device could be found matching the *device-url* that you used. Use the `-i` option to enumerate the devices available.

The message “Cannot force background mode” can occur if you connect at too high a speed. Try slowing the connection by increasing the `speed=` option in the device URL.

5.4.2. Installing P&E Drivers

On Windows, the P&E driver is installed by Sourcery G++ Lite. If the P&E driver installation fails (for example, with an error about missing files), it may mean that you already have another copy of the drivers previously installed on your computer. Note that P&E drivers are not removed automatically when uninstalling Sourcery G++ Lite; you must do that separately using `Add/Remove Software` from the Windows control panel.

To reinstall the drivers on Windows, follow these steps:

1. Complete the Sourcery G++ Lite installation.
2. Turn off your system and disconnect all P&E devices.
3. Reboot the system and use `Add/Remove Software`, available through the Windows control panel, to check for and remove any previously-installed P&E drivers.
4. Run `libexec/m68k-linux-gnu-post-install/sprite-drivers/pe_drivers_install.bat` to reinstall the drivers.
5. Turn off your system and connect all P&E devices.
6. Reboot the system and start using Sourcery G++ Lite.

On Linux, the P&E driver is a loadable kernel module that has to be compiled for your system. You need kernel headers and a native C compiler for your system. The package is `pe_driver_ver_324_811.tar.gz` and is in the `libexec/m68k-linux-gnu-post-install/sprite-drivers` subdirectory of your Sourcery G++ Lite installation. You should unpack that file, and use the `setup.sh` script to build and install it. You should manually remove all files of a previous install before building this module.

These drivers are provided by P&E Microcomputer Systems.

5.5. Command Converter Server Devices

The Sourcery G++ Debug Sprite supports devices such as the CodeWarrior Ethernet TAP and USB TAP that are controlled by the Command Converter Server (CCS) utility. You need to start CCS separately before connecting to the debug device from GDB; see Section 5.5.1, “Starting CCS”.

The Sprite partitions the CCS *`device-url`* as follows:

```
ccs:[//host[:port]][/chainpos][?device-options]
```

The *`host`* and *`port`* indicate the location of the CCS port to connect to. The *`chainpos`* (a number) indicates where the ColdFire debug device is in the CCS chain.

The following *`device-options`* are permitted:

`speed=`*`speed`* Specify the speed used to connect to the target. This is specified in KHz by default. You can use `MHz` and `KHz` suffixes.

`timeout=`*`timeout`* This specifies the timeout, in seconds, used for communication with the Command Converter Server.

As an example, if CCS is listening on port `localhost:41475`, connect GDB to the board with:

```
(gdb) target remote | \
  m68k-linux-gnu-sprite ccs://localhost:41475 m54455evb
```

5.5.1. Starting CCS

CCS is included with Sourcery G++ Lite; you do not need to have the CodeWarrior tools installed. You can find the CCS executable in the `m68k-linux-gnu/ccs/bin` subdirectory of your Sourcery G++ Lite installation.

The server can be started by clicking on the CCS icon, or by entering **ccs** on the command line. You can use the `-nogfx` option to use its command-line interface rather than having it create a GUI window.

Use the following commands to initialize the server:

```
% delete all
% config port port
% config cc device
% config client all
```

The *`port`* number is the TCP/IP port the server listens on, and is what you should use in the Sprite's URI. The *`device`* indicates what target device should be used. For USB devices use `utap` for COP/OnCE and `utap_dpi` for BDM or DPI. For Ethernet devices use `powertap` for COP/OnCE

and `powertap_dpi` for BDM or DPI. If you have multiple devices of you can append a :*`serial-number`* to the USB *`device`* name. The eight-digit *`serial-number`* is located on the underside of the TAP device just after the revision information. For Ethernet devices append the device's IP address.

In summary, to connect to a COP/OnCE target using an Ethernet TAP:

```
% config cc powertap:1.2.3.4
```

To connect to a BDM or DPI target using an Ethernet TAP:

```
% config cc powertap_dpi:1.2.3.4
```

To connect to a COP/OnCE target using a USB TAP:

```
% config cc utap
```

To connect to a BDM or DPI target using a USB TAP:

```
% config cc utap_dpi
```

You can use the **config save** command to save the configuration for later use. The **show cc** command shows you the current configuration. The **show port** command shows you the port number CCS is serving.

5.5.2. Common CCS Errors

Here are some common error messages and their causes:

Cable disconnected	The target board is not powered up, the board hardware is faulty or in a bad state or the jumper settings are incorrect.
CC not present	The required Command Converter is not present. You did not use `utap_bdm` or `utap_dpi` to connect CCS to a BDM or DPI TAP device connection.
Core not responding	CCS is no longer has control of the target system. The board hardware is faulty or in a bad state, the board initialization settings are incorrect or there is another debugger configuration problem.
USB open failure	For a Windows host, the USB driver on the host computer is hung. Unplug/replug the USB tap, or reboot the host PC if the problem persists. This might also happen if the USB drivers were not installed. You may install USB drivers manually from `m68k-linux-gnu\ccs\drivers` subdirectory of your Sourcery G++ Lite installation. For a Linux host this can occur if the permissions are not set correctly. Try running CCS as root, and if this resolves the problem, review the instructions in `m68k-linux-gnu/ccs/drivers/usb` subdirectory of your Sourcery G++ Lite installation for setting up USB permissions.
Maximum number of Command Converters reached	You have tried to reconfigure without first deleting the current configuration.

Cannot reset to debug mode	This can indicate that the clock speed is too high. Try a lower clock speed with the `speed=` option in the device URL.

5.6. Turbo BDM Light ColdFire Devices

Turbo BDM Light ColdFire (TBLCF) devices, such as the Axiom AxBDM device, are supported. The TBLCF device type partitions the *`device-url`* as follows:

```
tblcf:[//:number/]
```

The *`number`* indicates the number of the TBLCF interface to connect to, counting from zero upwards. If the number is omitted, the default is to connect to the zeroth interface, which works well if you have only one TBLCF device connected to your computer.

There are no further options for the TBLCF device.

If you are connecting via TBLCF from Windows, you may see a message like:

```
m68k-linux-gnu-sprite:error: Couldn't load libusb DLL
```

If this happens, you must install the driver for the TBLCF device, included with Sourcery G++ Lite. See Section 5.6.1, “Installing TBLCF (AxBDM) Windows Drivers” for installation instructions.

If you are connecting via TBLCF from Linux, you may see a message like:

```
m68k-linux-gnu-sprite:error: Error claiming interface \
(-1, permission denied)
```

If you see this message, consult Section 5.6.2, “Configuring TBLCF (AxBDM) Devices on Linux” for configuration instructions.

5.6.1. Installing TBLCF (AxBDM) Windows Drivers

Before using a TBLCF device, you must install a driver. To install the TBLCF (AxBDM) driver on Windows, follow these steps:

1. Complete the Sourcery G++ Lite installation.
2. Run the `Add Hardware` Control Panel. Click `Yes, I have already connected the hardware`.
3. Select `Add a new hardware device`.
4. Select `Install the hardware that I manually select from a list`.
5. Select `Show all devices`.
6. Click `Have Disk`. Browse to `libexec/m68k-linux-gnu-post-install/axbdm-drivers/axbdm.inf`, then select `AxBDM` from the list on the following pane.
7. You will get warnings about the driver not being signed by Microsoft. This is expected.
8. Reboot the system when prompted and start using Sourcery G++ Lite.

Windows may auto-detect the TBLCF device when it is connected, and invoke the driver installation dialog automatically. If you have already installed Sourcery G++ Lite, you may continue with the

dialog using steps similar to those outlined above. Otherwise, close the dialog, install Sourcery G++ Lite first, and then follow the above steps to install the driver.

5.6.2. Configuring TBLCF (AxBDM) Devices on Linux

The method you should use for configuring the TBLCF device on Linux depends on whether your machine is using udev or hotplug to manage USB device permissions. To determine which of these your distribution uses, find out your kernel and udev version numbers as follows:

```
> uname -r
2.6.20
> udevinfo -V
udevinfo, version 108
```

A rule of thumb is that if your kernel version is less than 2.6.13 (*2.6.20* in the example) or your udev version is less than 059 (*108* in the example), your machine uses hotplug to control USB device permissions, else it uses udev. If this rule of thumb doesn't work for you, consult your operating system vendor to determine which method your distribution uses.

Performing the following steps allows any user to access the TBLCF device, rather than just the superuser (root). Running the Debug Sprite as root is technically possible, but is *strongly* discouraged.

5.6.2.1. Configuring TBLCF with udev

To configure udev to handle TBLCF permissions, first locate your udev rule configuration directory (e.g. `/etc/udev/rules.d/`). As root, create a file in that directory called `25-tblcf.rules` with the following contents:

```
BUS=="usb", SYSFS{idVendor}=="0425", SYSFS{idProduct}=="1001", \
MODE="0666"
```

Note that this should be entered on one line, without the backslash. Once this file is created, plug in the TBLCF device (if it is not already plugged in) then reboot your machine to make sure your changes take effect.

5.6.2.2. Configuring TBLCF with hotplug

To configure hotplug to handle TBLCF permissions, you must create two files in your hotplug USB configuration directory (e.g. `/etc/hotplug/usb/`) as root. The first file is named `tblcf` and contains:

```
#!/bin/bash
# /etc/hotplug/usb/tblcf
#
if [ "${ACTION}" = "add" ] && [ -f "${DEVICE}" ]
then
    case "$PRODUCT" in
        425/1001/*)
            chmod 0666 "${DEVICE}"
            ;;
    esac
fi
```

The second file (in the same directory) is named `tblcf.usermap` and contains:

```
tblcf 0x0003 0x0425 0x1001 0x0000 0x0000 0x00 0x00 0x00 0x00 \
0x00 0x00 0x00000000
```

Note that the above must be entered on one line, without the backslash. Create these files and plug in your TBLCF device, if it is not already plugged in. Reboot your machine to make sure your changes take effect.

5.6.2.3. Troubleshooting TBLCF Device Permissions

If you are having difficulties using the Debug Sprite as a non-root user, check that your udev or hotplug configuration is working properly by ensuring that the TBLCF device has the right file permissions. To do this, first run the following command:

```
> lsusb -d 0x0425:0x1001
Bus 004 Device 002: ID 0425:1001 Motorola Semiconductors HK, Ltd
```

Note the bus and device number (*004* and *002* above). Now, examine the permissions of the corresponding device file as follows:

```
> ls -l /proc/bus/usb/004/002
-rw-rw-rw-  1 root root 50 2007-11-02 12:12 /proc/bus/usb/004/002
```

If the file has permissions as shown, you should be able run the Debug Sprite as any user, and the problem lies elsewhere. If the permissions are different, or there was no output from the **lsusb** command above, your configuration is not working properly. Ask CodeSourcery for further guidance.

5.7. Open Source BDM Devices

Open Source BDM (OSBDM) devices are supported. The OSBDM device type partitions the *device-url* as follows:

```
osbdm:[//number/]
```

The *number* indicates the number of the OSBDM interface to connect to, counting from zero upwards. If the number is omitted, the default is to connect to the zeroth interface, which works well if you have only one OSBDM device connected to your computer.

There are no further options for the OSBDM device.

If you are connecting via OSBDM from Windows, you may see a message like:

```
m68k-linux-gnu-sprite:error: Cannot load OSBDM library \
'OSBDM-JM60.DLL'
```

If this happens, you must install the driver for the OSBDM device. You can obtain the driver from the vendor of your OSBDM device.

As of this writing, there is not yet an OSBDM driver available for Linux hosts.

5.8. Debugging a Remote Board

You can run the Sourcery G++ 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 G++ Debug Sprite must run on the

machine that is connected to the target board. You must have Sourcery G++ 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:

```
> m68k-linux-gnu-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 G++ Debug Sprite in this way, please refer to the Sourcery G++ Knowledge Base[1].

5.9. Implementation Details

The Sourcery G++ Debug Sprite uses Background Debug Mode, which is supported by all ColdFire cores. In most cases this is completely non-intrusive to the program being debugged. However, if you are using the Sourcery G++ Debug Sprite to debug an operating system kernel (or program with kernel-like features), some of the debugging operations can interact with the program being debugged.

5.9.1. Software Breakpoints

The Debug Sprite uses `HALT` instructions to implement software breakpoints and semihosting. On execution of a `HALT` instruction, the Debug Sprite gains control. If the `HALT` instruction is one that the Debug Sprite inserted itself, it reports a breakpoint to the host's GDB. Semihosting breakpoints are detected by checking for the bit pattern `0x4e7bf000`, which corresponds to an unrealistic `movec %sp,0` instruction. The semihosting operation will be performed and the program counter adjusted to skip the ill-formed instruction. For all other `HALT` instructions GDB will report a `SIGTRAP`.

If the program being debugged uses `HALT` instructions in an idle loop, each iteration of the idle loop will cause such a `SIGTRAP` to be reported by GDB. If you want GDB to ignore these signals, enter the following GDB command:

```
handle SIGTRAP nostop noprint nopass
```

As `HALT` is a privileged instruction, the Debug Sprite sets the `UHE` bit in the `CSR` so that user mode programs do not raise a privilege violation exception on `HALT` execution.

5.9.2. Hardware Watchpoints

A single hardware watchpoint is implemented using ColdFire's `TDR`, `AATR`, `ABLR` & `ABHR` debug registers (Trigger Definition Register, Address Attribute Trigger Register, Address Bus Low Register and Address Bus High Register respectively). A range of addresses can watch for data read, write or access.

[1] https://support.codesourcery.com/GNUToolchain/kbentry132

Because of the way ColdFire implements the address range check, it is possible for an access to an address just before the range, but whose final byte is within the watched range to be undetected. For instance watching a single byte at address `4N+3` fails to trigger on 32 bit writes to address `4N` or on 16 bit writes to address `4N+2`.

5.9.3. Single Stepping

Single stepping uses the ColdFire single step feature. This is performed with the `IPI` (Ignore Pending Interrupts) bit set in the `CSR`. Without this bit set, single stepping an instruction when an interrupt is pending stops at the first instruction of the ISR, which is undesirable. Thus single stepping a sequence of instructions does not process any interrupts. During continuous execution, interrupts are not so inhibited, and ISRs are executed, if the remainder of the processor state allows them. GDB commands that perform single stepping are **step** and **stepi**. Commands that perform continuous execution are **continue**, **jump** and **finish**. The **next** and **nexti** commands perform single stepping, except when a function is called, in which case they perform a sequence of single steps to enter the called function, followed by continuous execution for the bulk of the called function.

5.10. Supported Board Files

The Sourcery G++ Debug Sprite for ColdFire GNU/Linux 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**
Freescale M54455EVB	`m54455evb`
Freescale M54455EVB (Intel flash at CS0)	`m54455evb-intel`
Freescale M5485EVB	`m5485evb`

5.11. Board File Syntax

The *`board-file`* can be a user-written XML file to describe a non-standard board. The Sourcery G++ Debug Sprite searches for board files in the `m68k-linux-gnu/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) 2007-2009 CodeSourcery, Inc.

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

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

<!ELEMENT board
 (properties?, feature?, initialize?, memory-map?)>
```

```
<!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>

<!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 `<properties>`, `<feature>`, `<initialize>` and `<memory-map>` elements.

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

It can also contain `<property>` elements with the following names:

`cache` This boolean property is used to indicate that the target has a cache. This knowledge is necessary to correctly write to a program's instruction stream.

`floating-point` This boolean property indicates whether floating point registers are provided on the target.

`<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.

It can also contain the following named `<property>` elements for additional flash-specific information:

`system-clock` This numeric property is used for `cfm` flash devices. It specifies the target frequency, and is used to determine the flash frequency divider value.

`page-size` This numeric property is used for `cfm` flash devices. It specifies the flash logical page size. When not specified, the page size is set to 1K for the ColdFire V1 devices and to 2K for the ColdFire V2+ devices.

`<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. The address may be specified as a number, or as a name. The following registers are

	available: `ASID`, `ACR0`, `ACR1`, `ACR1`, `ACR1`, `MMUBAR`, `VBR`, `ROMBAR0`, `ROMBAR1`, `FLASHBAR`, `RAMBAR0`, `RAMBAR1`, `MPCR`, `EDRAMBAR`, `SECMBAR`, `MBAR2`, `MBAR`.
`<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.

Chapter 6
Next Steps with Sourcery G++

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

6.1. Sourcery G++ Subscriptions

CodeSourcery offers two levels of Sourcery G++ subscriptions. Professional Edition subscriptions include unlimited support, with no per-incident fees. CodeSourcery's support is provided by the same engineers who build Sourcery G++, and covers questions about installing and using Sourcery G++, the C and C++ programming languages, and all other topics relating to Sourcery G++. CodeSourcery provides updated versions of Sourcery G++ on demand to resolve critical problems reported by Professional Edition subscribers. Personal Edition subscriptions do not include support, but do include access to updates as long as the subscription remains active.

Subscription editions of Sourcery G++ also include many additional features not included in the free Lite editions:

- **Sourcery G++ IDE.** The Sourcery G++ IDE, based on Eclipse, provides a fully visual environment for developing applications, including an automated project builder, syntax-highlighting editor, and a graphical debugging interface. The debugger provides features especially useful to embedded systems programmers, including the ability to step through code at both the source and assembly level, view registers, and examine stack traces. CodeSourcery's enhancements to Eclipse include improved support for hardware debugging via JTAG or ICE units and complete integration with the rest of Sourcery G++.

- **Debug Sprites.** Sourcery G++ Debug Sprites provide hardware debugging support using JTAG and ICE devices. On some systems, Sourcery G++ Sprites can automatically program flash memory and display control registers. Debug Sprites included in Lite editions of Sourcery G++ include only a subset of the functionality of the Sprites in the subscription editions.

- **CS3.** CS3 provides a uniform, cross-platform approach to board initialization and interrupt handling on bare-metal ELF and EABI platforms. Subscription versions of Sourcery G++ include CS3 support for an expanded set of boards. In addition, the Sourcery G++ Board Builder allows you to extend the power of CS3 to cover custom board definitions. The Board Builder is fully integrated with the Sourcery G++ IDE and Debug Sprites.

- **CodeSourcery C Library.** Subscription versions of Sourcery G++ for bare-metal targets include the CodeSourcery C Library, a proprietary library implementation that is optimized to be smaller and faster than the Newlib C library included with Lite editions of Sourcery G++.

- **QEMU Instruction Set Simulator.** The QEMU instruction set simulator can be used to run — and debug — programs even without target hardware. Most bare-metal configurations of Sourcery G++ include QEMU and linker scripts targeting the simulator. Configurations of Sourcery G++ for GNU/Linux targets include a user-space QEMU emulator that runs on Linux hosts.

- **Sysroot Utilities.** Subscription editions of Sourcery G++ include a set of sysroot utilities for GNU/Linux targets. These utilities simplify use of the Sourcery G++ dynamic linker and shared libraries on the target and also support remote debugging with **gdbserver**.

- **GNU/Linux Prelinker.** For select GNU/Linux target systems, Sourcery G++ includes the GNU/Linux prelinker. The prelinker is a postprocessor for GNU/Linux applications which can dramatically reduce application launch time. CodeSourcery has modified the prelinker to operate on non-GNU/Linux host systems, including Microsoft Windows.

- **Library Reduction Utility.** Sourcery G++ also includes a Library Reduction Utility for GNU/Linux targets. This utility allows the GNU C Library to be relinked to include only those functions used by a given collection of binaries.

- **Additional Libraries.** For some platforms, additional run-time libraries optimized for particular CPUs are available. Pre-built binary versions of the libraries with debug information are also available to subscribers.

- **Additional Documentation.** Subscription customers receive expanded access to the Sourcery G++ Knowledge Base, covering many more tips, howtos, and application notes to help you make the best use of Sourcery G++.

If you would like more information about Sourcery G++ subscriptions, including a price quote or information about evaluating Sourcery G++, please send email to <sales@codesourcery.com>.

If you have a Sourcery G++ subscription, you may access your account by visiting the Sourcery G++ Portal[1]. If you have a support account, but are unable to log in, send email to <support@codesourcery.com>.

6.2. Sourcery G++ Knowledge Base

The Sourcery G++ Knowledge Base is available to registered users at the Sourcery G++ Portal[2]. Here you can find solutions to common problems including installing Sourcery G++, 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.

6.3. Manuals for GNU Toolchain Components

Sourcery G++ 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 G++ 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 G++ Lite, the documentation can be found in the `share/doc/sourceryg++-m68k-linux-gnu/` subdirectory of your installation directory.

In addition to the detailed reference manuals, Sourcery G++ 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/sourceryg++-m68k-linux-gnu/man/man1
```

Then you can invoke **man** as:

```
> man ./m68k-linux-gnu-gcc.1
```

Alternatively, if you use **man** regularly, you'll probably find it more convenient to add the directory containing the Sourcery G++ man pages to your `MANPATH` environment variable. This should go in your `.profile` or equivalent shell startup file; see Section 2.6, “Setting up the Environment” for instructions. Then you can invoke **man** with just the command name rather than a pathname.

[1] https://support.codesourcery.com/GNUToolchain/
[2] https://support.codesourcery.com/GNUToolchain/

Finally, note that every command-line utility program included with Sourcery G++ 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.

Appendix A
Sourcery G++ Lite Release Notes

This appendix contains information about changes in this release of Sourcery G++ Lite for ColdFire GNU/Linux. You should read through these notes to learn about new features and bug fixes.

A.1. Changes in Sourcery G++ Lite for ColdFire GNU/Linux

This section documents Sourcery G++ Lite changes for each released revision.

A.1.1. Changes in Sourcery G++ Lite 4.4-54

gdbserver bug fix. A bug has been fixed that caused **gdbserver** to crash when debugging programs using thread-local storage without other multi-threading features.

`@FILE` fix. A bug has been fixed in the processing of `@FILE` command-line options by GCC, GDB, and other tools. The bug caused any options in `FILE` following a blank line to be ignored.

Preprocessor error handling. The preprocessor now treats failing to find a file referenced via `#include` as a fatal error.

Stack unwinding bug fix. A compiler bug that resulted in incorrect stack unwinding information has been fixed. The bug interfered with the **next** and **finish** commands when debugging, and could cause programs using C++ exception handling to crash.

ELF file corruption with strip. A bug that caused **strip** to corrupt unusual ELF files has been fixed.

GDB support for Cygwin pathnames. A bug in GDB's translation of Cygwin pathnames has been fixed.

gdbserver multi-threaded debugging fix. A bug has been fixed that prevented **gdbserver** from exiting after debugging a multi-threaded program.

GCC internal compiler error. A bug has been fixed that caused the compiler to crash when optimizing code that casts between structure types and the type of the first field.

ELF Program Headers. The linker now better diagnoses errors in the usage of `FILEHDR` and `PHDRS` keywords in `PHDRS` command of linker scripts. Refer to the linker manual for more information.

A.1.2. Changes in Sourcery G++ Lite 4.4-29

ColdFire MCF5441x support. Sourcery G++ now supports the Freescale MCF5441x (Modelo) family of microprocessors. To compile for these CPUs use the `-mcpu=54410`, `-mcpu=54415`, `-mcpu=54416`, `-mcpu=54417`, and `-mcpu=54418` command-line options.

Linux kernel headers update. Linux kernel header files have been updated to version 2.6.30. Among other things this fixes assembler failures with functions that convert integers to different endianness.

Optimizer improvements. When optimizing for speed, the compiler now uses improved heuristics to limit certain types of optimizations that may adversely affect both code size and speed. This change also makes it possible to produce better code when optimizing for space rather than speed.

EGLIBC version 2.10. Sourcery G++ Lite for ColdFire GNU/Linux now includes the EGLIBC version 2.10 library, based on GNU C Library version 2.10. This is a major upgrade from the version 2.5 included in previous releases of Sourcery G++, and brings GLIBC support up to date on the ColdFire platform. New features introduced by this upgrade include support for NPTL (Native POSIX

Thread Library) and related features such as thread-local storage. For more information about other EGLIBC changes, see `http://www.eglibc.org/news#eglibc_2_10`.

Linux kernel update required. As a consequence of upgrading to a more current version of EGLIBC, Sourcery G++ Lite now requires more recent Linux kernel versions and an additional kernel patch. Refer to Section 3.3, "Target Kernel Requirements" for more information.

GDB update. The included version of GDB has been updated to 6.8.50.20090630. This update adds numerous bug fixes and new features, including support for multi-byte and wide character sets and improved C++ template support.

GDB and third-party compilers. Some bugs that caused GDB to crash when debugging programs compiled with third-party tools have been fixed. These bugs did not affect programs built with Sourcery G++.

Remote debugging hardware watchpoint bug fix. A GDB bug has been fixed that caused hardware watchpoint hits to be incorrectly reported in some cases.

GDB internal warning fix. A GDB bug has been fixed that caused warnings of the form `warning: (Internal error: pc` *`address`* `in read in psymtab, but not in symtab.)`.

Binutils update. The binutils package has been updated to version 2.19.51.20090709 from the FSF trunk. This update includes numerous bug fixes.

Code generation for string literals. A defect in the generation of code for string literals has been corrected. Multiple occurrences of the same string literal in the same file sometimes resulted in incorrect code.

Configuration file required for Debug Sprite. When invoking the Sourcery G++ Debug Sprite from the command line, it is now required to specify a board configuration file argument. This change eliminates a source of confusion and errors resulting from accidental omission of the configuration file argument, since recent improvements to debugger functionality depend on properties specified in the configuration file. Refer to Chapter 5, "Sourcery G++ Debug Sprite" for more details on invoking the Sourcery G++ Debug Sprite from the command line.

GDB segmentation fault bug fix. A bug in the Sourcery G++ Debug Sprite that sometimes caused GDB to crash when inspecting register contents has been fixed.

Register variable corruption. A compiler bug has been fixed that caused incorrect code to be generated when the frame pointer or other special-use registers are used as explicit local register variables, introduced via the `asm` keyword on their declarations.

Startup code debugging fixes. Two GDB bugs have been fixed that caused errors when debugging startup code. One bug caused an internal error message; the other caused the error `Cannot find bounds of current function`.

`-fremove-local-statics` optimization. The `-fremove-local-statics` optimization is now enabled by default at `-O2` and higher optimization levels.

Elimination of spurious warnings about `NULL`. The C++ compiler no longer issues spurious warnings about comparisons between pointers to members and `NULL`.

Vectorizer improvements. The compiler now generates improved code for accesses to static nested array variables (e.g. `static int foo[8][8];`).

GCC version 4.4.1. Sourcery G++ Lite for ColdFire GNU/Linux is now based on GCC version 4.4.1. For more information about changes from GCC version 4.3 that was included in previous releases, see `http://gcc.gnu.org/gcc-4.4/changes.html`.

Linker map address sorting. The map generated by the linker `-Map` option now lists symbols sorted by address.

A.1.3. Changes in Sourcery G++ Lite 4.3-210

GDB finish internal error. A bug has been fixed that caused a GDB internal error when using the **finish** command. The bug occurred when debugging optimized code.

GCC version 4.3.3. Sourcery G++ Lite for ColdFire GNU/Linux is now based on GCC version 4.3.3. This is a bug fix update to GCC. For more information about changes from GCC version 4.3.2 that was included in previous releases, see `http://gcc.gnu.org/gcc-4.3/changes.html`.

Bug fix for assembly listing. A bug that caused the assembler to produce corrupted listings (via the `-a` option) on Windows hosts has been fixed.

Incorrect code when using `-falign-labels`. A bug that caused the compiler to generate incorrect code for `switch` statements when the `-falign-labels` option is used has been fixed.

Reduced compilation time. Compilation and build times when using Sourcery G++ Lite are now slightly faster. This performance improvement is the result of building the compilers and other host tools with a recent version of Sourcery G++, rather than an older GCC version.

Internal compiler error with `-O3` or `-fpredictive-commoning`. A bug has been fixed that caused internal compiler errors when compiling some code with `-O3` or `-fpredictive-commoning`.

C++ named operators bug fix. A bug has been fixed that caused the compiler to crash in some cases when the C++ operators `and_eq`, `bitand`, `bitor`, `compl`, `not_eq`, `or_eq` and `xor_eq` were used in contexts where the preprocessor converts their names to strings.

Debug information for anonymous structure types. A GCC bug in the generation of debug information for anonymous structure types in C++ code has been fixed. The bug caused printing the type information for such structures in the debugger (via the **ptype** command) to fail with an error message.

GDB display of source. A bug has been fixed that prevented GDB from locating debug information in some cases. The debugger failed to display source code for or step into the affected functions.

Sprite's failure to reset the target. A bug has been fixed that sometimes caused the Sourcery G++ Debug Sprite to fail to reset the target when using the multiple sequential connection feature (enabled via the `-m` command-line option). This problem was specific to running the Debug Sprite on Microsoft Windows hosts.

Installer fails during upgrade. The Sourcery G++ installer for Microsoft Windows hosts could fail during an upgrade while waiting for the previous version to be uninstalled. This bug has been fixed.

Loop optimization improvements. A new option, `-fpromote-loop-indices`, has been added to the compiler. Specifying this option enables an optimization that improves the performance of loops with index variables of integer types narrower than the target machine word size, such as `char` or `short`. This optimization also applies to `int` on 64-bit targets.

Overloaded function resolution. The C++ compiler now correctly diagnoses an error when the second operand of a comma expression is an unresolved set of overloaded functions. Previously, it incorrectly used the context of the comma expression to resolve the function.

Uninstaller removed by upgrade. The uninstaller could be incorrectly deleted during an upgrade on Microsoft Windows hosts. This bug has been fixed.

Remote debugging connection auto-retry. The **target remote** command within GDB now uses a configurable auto-retry timeout when establishing TCP connections. This is useful in avoiding race conditions when the remote GDB stub or GDB server is launched simultaneously with GDB. The auto-retry behavior is enabled by default; refer to the GDB manual for details.

Extraneous linker error messages. A linker bug that caused extraneous error messages of the form `Dwarf Error: Offset (507) greater than or equal to .debug_str size (421).` has been corrected. This bug did not affect the correctness of output binaries.

GDB segment warning. Some compilers produce binaries including uninitialized data regions, such as the stack and heap. GDB incorrectly displayed the warning `Loadable segment "`*`name`*`" outside of ELF segments` for such binaries; the warning has now been fixed.

Internal compiler errors when optimizing. A defect that occasionally caused internal compiler errors when partial redundancy elimination (PRE) optimization was enabled has been corrected.

m68k-linux-gnu-objcopy bug fix. A bug has been fixed that caused **m68k-linux-gnu-objcopy** to issue an error when generating output in the Intel HEX format and using `--change-section-lma` to change section addresses.

Linker script search path. The bug in the linker has been fixed that caused it not to follow its documented behavior for searching for linker scripts named with the `-T` option. Now scripts are looked up first in the current directory, then in library directories specified with `-L` command-line options, and finally in the default system linker script directory.

Sprite crash on error. A bug has been fixed which sometimes caused the Sourcery G++ Debug Sprite to crash when it attempted to send an error message to GDB.

Errors when inserting breakpoints. A GDB bug has been fixed that caused errors of the form `` `*function*' found in *filename* psymtab but not in symtab `` when setting a breakpoint on *`function`*. This error commonly occurred when setting breakpoints on functions provided by the C library.

Install directory pathnames. Bugs in the install and uninstall scripts for Linux hosts that caused errors or incorrect behavior when the Sourcery G++ install directory pathname contains whitespace characters have been fixed.

Temporary files on Microsoft Windows. On Microsoft Windows hosts, Sourcery G++ Lite now uses the standard Windows algorithm to choose the directory in which to place temporary files. This change eliminates a crash that occurred if none of the `TEMP`, `TMP`, or `TMPDIR` variables were set to a suitable directory.

Open Source BDM ColdFire support. Initial support for Open Source BDM (OSBDM) probes has been added to the Sourcery G++ Debug Sprite. Both integrated and stand-alone OSBDM probes are supported. Using this feature requires installed OSBDM drivers, which are currently only available for Windows hosts. For more information, see Section 5.7, “Open Source BDM Devices”.

Internal compiler error when optimizing. A bug has been fixed that caused `internal compiler error: in build2_stat` when compiling.

Binutils update. The binutils package has been updated to version 2.19.51.20090205 from the FSF trunk. This update includes numerous bug fixes.

GDB quit error. A bug in GDB has been fixed that caused **quit** to report `Quitting: You can't do that without a process to debug.` when debugging a core dump file.

Internal compiler error with `-fremove-local-statics`. An internal compiler error that occurred when using the `-fremove-local-statics` option has been fixed. The error occurred when compiling code with function-local `static` array or structure variables.

Corruption of block-scope variables. A compiler optimization bug that sometimes caused corruption of stack-allocated variables has been fixed. The bug affected variables declared in a local block scope in functions containing multiple non-overlapping lexical block scopes, a technique commonly used by programmers to reduce stack frame size. In some rare cases, other optimizations performed by the compiler were ignoring the local extent of such block-scope variables.

Persistent remote server connections. A GDB bug has been fixed that caused the **target extended-remote** command to fail to tell the remote server to make the connection persistent across program invocations.

GDB update. The included version of GDB has been updated to 6.8.50.20081022. This update includes numerous bug fixes.

Pointer-to-member functions. A bug has been fixed that caused the C++ compiler to crash when compiling a pointer-to-member function reference without an explicit & operator. This syntax is allowed only when the `-fms-extensions` command-line option is used.

A.1.4. Changes in Sourcery G++ Lite 4.3-43

Printing casted values in GDB. A GDB bug that caused incorrect output for expressions containing casts, such as in the `print *(Type *)ptr` command, has been fixed.

Bug fix for objcopy/strip. An objcopy bug that corrupted COMDAT groups when creating new binaries has been fixed. This bug also affected **strip -g**.

Binutils support for DWARF Version 3. The **addr2line** command now supports binaries containing DWARF 3 debugging information. The **ld** command can display error messages with source locations for input files containing DWARF 3 debugging information.

P&E driver updates. The P&E drivers for Windows and Linux have been updated to version 3.32-920.

Connecting to the target using a pipe. A bug in GDB's **target remote | *`program`*** command has been fixed. When launching the specified *`program`* failed, the bug caused GDB to crash, hang, or give a message `Error: No Error`.

Modifying control registers. A bug has been fixed which prevented writes to processor and device control registers when using the Sourcery G++ Debug Sprite.

P&E ColdFire V1 support. The P&E drivers now include support for V1 ColdFire devices.

Code generation bug fix. A bug has been fixed that caused the compiler to generate invalid code which was rejected by the assembler with an `operands mismatch` error.

Errors after loading the debugged program. An intermittent GDB bug has been fixed. The bug could cause a GDB internal error after the **load** command.

A.1.5. Changes in Sourcery G++ Lite 4.3-11

GDB update. The included version of GDB has been updated to 6.8.50.20080821. This update adds numerous bug fixes and new features, including support for decimal floating point, the new **find** command to search memory, the new `/m` (mixed source and assembly) option to the **disassemble** command, and the new **macro define** command to define C preprocessor macros interactively.

Remote debugging improvements. The **gdbserver** utility now supports a more efficient communications protocol that can reduce latency during remote debugging. The protocol optimizations are enabled automatically when **gdbserver** operates over a TCP connection. Refer to the GDB manual for more information.

Output files removed on error. When GCC encounters an error, it now consistently removes any incomplete output files that it may have created.

Cache control. A bug in the Debug Sprite has been fixed that previously caused failures when stepping over breakpoints on V3, V4 and V4e cores when caching is enabled.

Processor status register display. When displaying the status register, both GDB and the Sourcery G++ IDE now list the names of flags that are set. Previously, a decimal number was displayed.

GCC version 4.3.2. Sourcery G++ Lite for ColdFire GNU/Linux is now based on GCC version 4.3.2. For more information about changes from GCC version 4.2 that was included in previous releases, see `http://gcc.gnu.org/gcc-4.3/changes.html`.

Sprite communication improvements. The Sourcery G++ Debug Sprite now uses a more efficient protocol for communicating with GDB. This can result in less latency when debugging, especially when running the Sprite on a remote machine over a network connection.

Bug fix for objdump on Windows. An objdump bug that caused the `-S` option not to work on Windows in some cases has been fixed.

A.1.6. Changes in Older Releases

For information about changes in older releases of Sourcery G++ Lite for ColdFire GNU/Linux, please refer to the Getting Started guide packaged with those releases.

Appendix B
Sourcery G++ Lite Licenses

Sourcery G++ 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 G++ Lite. You should read this appendix to understand your legal rights and obligations as a user of Sourcery G++ Lite.

B.1. Licenses for Sourcery G++ Lite Components

The table below lists the major components of Sourcery G++ Lite for ColdFire GNU/Linux and the license terms which apply to each of these 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 G++ Lite. Sourcery G++ 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 G++ Lite.

Component	**License**
GNU Compiler Collection	GNU General Public License 3.0 [1]
GNU Binary Utilities	GNU General Public License 3.0 [2]
GNU Debugger	GNU General Public License 3.0 [3]
Sourcery G++ Debug Sprite for ColdFire	CodeSourcery License
CCS Server	CCS Server License
GNU C Library	GNU Lesser General Public License 2.1 [4]
Linux Kernel Headers	GNU General Public License 2.0 [5]
GNU Make	GNU General Public License 2.0 [6]
GNU Core Utilities	GNU General Public License 2.0 [7]

The CodeSourcery License is available in Section B.2, “Sourcery G++ Software License Agreement”.

Important

Although some of the licenses that apply to Sourcery G++ 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 G++ Lite. You can develop proprietary applications and libraries with Sourcery G++ Lite.

Sourcery G++ Lite may include some third party example programs and libraries in the `share/sourceryg++-m68k-linux-gnu-examples` subdirectory. These examples are not covered by the Sourcery G++ 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.

[1] http://www.gnu.org/licenses/gpl.html
[2] http://www.gnu.org/licenses/gpl.html
[3] http://www.gnu.org/licenses/gpl.html
[4] http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html
[5] http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
[6] http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
[7] http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

B.2. Sourcery G++™ Software License Agreement

1. **Parties.** The parties to this Agreement are you, the licensee ("You" or "Licensee") and CodeSourcery. If You are not acting on behalf of Yourself as an individual, then "You" means Your company or organization.

2. **The Software.** The Software licensed under this Agreement consists of computer programs and documentation referred to as Sourcery G++™ Lite Edition (the "Software").

3. **Definitions.**

    3.1. **CodeSourcery Proprietary Components.** The components of the Software that are owned and/or licensed by CodeSourcery and are not subject to a "free software" or "open source" license, such as the GNU Public License. The CodeSourcery Proprietary Components of the Software include, without limitation, the Sourcery G++ Installer, any Sourcery G++ Eclipse plug-ins, and any Sourcery G++ Debug Sprite. For a complete list, refer to the *Getting Started Guide* included with the distribution.

    3.2. **Open Source Software Components.** The components of the Software that are subject to a "free software" or "open source" license, such as the GNU Public License.

    3.3. **Proprietary Rights.** All rights in and to copyrights, rights to register copyrights, trade secrets, inventions, patents, patent rights, trademarks, trademark rights, confidential and proprietary information protected under contract or otherwise under law, and other similar rights or interests in intellectual or industrial property.

    3.4. **Redistributable Components.** The CodeSourcery Proprietary Components that are intended to be incorporated or linked into Licensee object code developed with the Software. The Redistributable Components of the Software include, without limitation, the CSLIBC run-time library and the CodeSourcery Common Startup Code Sequence (CS3). For a complete list, refer to the *Getting Started Guide* included with the distribution.

4. **License Grant to Proprietary Components of the Software.** You are granted a non-exclusive, royalty-free license (a) to install and use the CodeSourcery Proprietary Components of the Software, (b) to transmit the CodeSourcery Proprietary Components over an internal computer network, (c) to copy the CodeSourcery Proprietary Components for Your internal use only, and (d) to distribute the Redistributable Component(s) in binary form only and only as part of Licensee object code developed with the Software that provides substantially different functionality than the Redistributable Component(s).

5. **Restrictions.** You may not: (i) copy or permit others to use the CodeSourcery Proprietary Components of the Software, except as expressly provided above; (ii) distribute the CodeSourcery Proprietary Components of the Software to any third party, except as expressly provided above; or (iii) reverse engineer, decompile, or disassemble the CodeSourcery Proprietary Components of the Software, except to the extent this restriction is expressly prohibited by applicable law.

    5.1. **Sourcery G++ Debug Sprite for P&E Devices.** You may use the Sourcery G++ Debug Sprite for P&E only in conjunction with ColdFire microprocessors and with debugging devices produced by P&E Microcomputer Systems.

    5.2. **Sourcery G++ Debug Sprite for CCS Debugging Devices.** The Sourcery G++ Debug Sprite for CCS includes the CodeWarrior Connection Server Dynamic Linked

Library ("CCS DLL") from Freescale Semiconductor, Inc. You may use the CCS DLL only in conjunction with Sourcery G++ on a Windows or Linux-hosted platform. You may not translate, reverse engineer, decompile, or disassemble the CCS DLL, except to the extent applicable law specifically prohibits such restriction. If You are a U.S. Government end user, the CCS DLL is "restricted computer software" and is subject to FAR 52.227-19(c)(1) and (c)(2).

6. **"Free Software" or "Open Source" License to Certain Components of the Software.** This Agreement does not limit Your rights under, or grant You rights that supersede, the license terms of any Open Source Software Component delivered to You by CodeSourcery. Sourcery G++ includes components provided under various different licenses. The *Getting Started Guide* provides an overview of which license applies to different components. Definitive licensing information for each "free software" or "open source" component is available in the relevant source file.

7. **CodeSourcery Trademarks.** Notwithstanding any provision in a "free software" or "open source" license agreement applicable to a component of the Software that permits You to distribute such component to a third party in source or binary form, You may not use any CodeSourcery trademark, whether registered or unregistered, including without limitation, CodeSourcery™, Sourcery G++™, the CodeSourcery crystal ball logo, or the Sourcery G++ splash screen, or any confusingly similar mark, in connection with such distribution, and You may not recompile the Open Source Software Components with the `--with-pkgversion` or `--with-bugurl` configuration options that embed CodeSourcery trademarks in the resulting binary.

8. **Term and Termination.** This Agreement shall remain in effect unless terminated pursuant to this provision. CodeSourcery may terminate this Agreement upon seven (7) days written notice of a material breach of this Agreement if such breach is not cured; provided that the unauthorized use, copying, or distribution of the CodeSourcery Proprietary Components of the Software will be deemed a material breach that cannot be cured.

9. **Transfers.** You may not transfer any rights under this Agreement without the prior written consent of CodeSourcery, which consent shall not be unreasonably withheld. A condition to any transfer or assignment shall be that the recipient agrees to the terms of this Agreement. Any attempted transfer or assignment in violation of this provision shall be null and void.

10. **Ownership.** CodeSourcery owns and/or has licensed the CodeSourcery Proprietary Components of the Software and all intellectual property rights embodied therein, including copyrights and valuable trade secrets embodied in its design and coding methodology. The CodeSourcery Proprietary Components of the Software are protected by United States copyright laws and international treaty provisions. CodeSourcery also owns all rights, title and interest in and with respect to its trade names, domain names, trade dress, logos, trademarks, service marks, and other similar rights or interests in intellectual property. This Agreement provides You only a limited use license, and no ownership of any intellectual property.

11. **Warranty Disclaimer; Limitation of Liability.** CODESOURCERY AND ITS LICENSORS PROVIDE THE SOFTWARE "AS-IS" AND PROVIDED WITH ALL FAULTS. CODESOURCERY DOES NOT MAKE ANY WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. CODESOURCERY SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SYSTEM INTEGRATION, AND DATA ACCURACY. THERE IS NO WARRANTY OR GUARANTEE THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED, ERROR-FREE, OR VIRUS-FREE, OR THAT THE SOFTWARE WILL MEET ANY PARTICULAR CRITERIA OF PERFORMANCE, QUALITY, ACCURACY,

PURPOSE, OR NEED. YOU ASSUME THE ENTIRE RISK OF SELECTION, INSTALLATION, AND USE OF THE SOFTWARE. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS AGREEMENT. NO USE OF THE SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

12. **Local Law.** If implied warranties may not be disclaimed under applicable law, then ANY IMPLIED WARRANTIES ARE LIMITED IN DURATION TO THE PERIOD REQUIRED BY APPLICABLE LAW.

13. **Limitation of Liability.** INDEPENDENT OF THE FORGOING PROVISIONS, IN NO EVENT AND UNDER NO LEGAL THEORY, INCLUDING WITHOUT LIMITATION, TORT, CONTRACT, OR STRICT PRODUCTS LIABILITY, SHALL CODESOURCERY BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND, INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER MALFUNCTION, OR ANY OTHER KIND OF COMMERCIAL DAMAGE, EVEN IF CODESOURCERY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY TO THE EXTENT PROHIBITED BY APPLICABLE LAW. IN NO EVENT SHALL CODESOURCERY'S LIABILITY FOR ACTUAL DAMAGES FOR ANY CAUSE WHATSOEVER, AND REGARDLESS OF THE FORM OF ACTION, EXCEED THE AMOUNT PAID BY YOU IN FEES UNDER THIS AGREEMENT DURING THE PREVIOUS ONE YEAR PERIOD.

14. **Export Controls.** You agree to comply with all export laws and restrictions and regulations of the United States or foreign agencies or authorities, and not to export or re-export the Software or any direct product thereof in violation of any such restrictions, laws or regulations, or without all necessary approvals. As applicable, each party shall obtain and bear all expenses relating to any necessary licenses and/or exemptions with respect to its own export of the Software from the U.S. Neither the Software nor the underlying information or technology may be electronically transmitted or otherwise exported or re-exported (i) into Cuba, Iran, Iraq, Libya, North Korea, Sudan, Syria or any other country subject to U.S. trade sanctions covering the Software, to individuals or entities controlled by such countries, or to nationals or residents of such countries other than nationals who are lawfully admitted permanent residents of countries not subject to such sanctions; or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nationals and Blocked Persons or the U.S. Commerce Department's Table of Denial Orders. By downloading or using the Software, Licensee agrees to the foregoing and represents and warrants that it complies with these conditions.

15. **U.S. Government End-Users.** The Software is a “commercial item,” as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of “commercial computer software” and “commercial computer software documentation,” as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire the Software with only those rights set forth herein.

16. **Licensee Outside The U.S.** If You are located outside the U.S., then the following provisions shall apply: (i) Les parties aux presentes confirment leur volonte que cette convention de meme que tous les documents y compris tout avis qui siy rattache, soient rediges en langue anglaise (translation: “The parties confirm that this Agreement and all related documentation is and will be in the English language.”); and (ii) You are responsible for complying with any local laws in your jurisdiction which might impact your right to import, export or use the Software, and You represent that You have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

17. **Severability.** If any provision of this Agreement is declared invalid or unenforceable, such provision shall be deemed modified to the extent necessary and possible to render it valid and enforceable. In any event, the unenforceability or invalidity of any provision shall not affect any other provision of this Agreement, and this Agreement shall continue in full force and effect, and be construed and enforced, as if such provision had not been included, or had been modified as above provided, as the case may be.

18. **Arbitration.** Except for actions to protect intellectual property rights and to enforce an arbitrator's decision hereunder, all disputes, controversies, or claims arising out of or relating to this Agreement or a breach thereof shall be submitted to and finally resolved by arbitration under the rules of the American Arbitration Association ("AAA") then in effect. There shall be one arbitrator, and such arbitrator shall be chosen by mutual agreement of the parties in accordance with AAA rules. The arbitration shall take place in Granite Bay, California, and may be conducted by telephone or online. The arbitrator shall apply the laws of the State of California, USA to all issues in dispute. The controversy or claim shall be arbitrated on an individual basis, and shall not be consolidated in any arbitration with any claim or controversy of any other party. The findings of the arbitrator shall be final and binding on the parties, and may be entered in any court of competent jurisdiction for enforcement. Enforcements of any award or judgment shall be governed by the United Nations Convention on the Recognition and Enforcement of Foreign Arbitral Awards. Should either party file an action contrary to this provision, the other party may recover attorney's fees and costs up to $1000.00.

19. **Jurisdiction And Venue.** The courts of Placer County in the State of California, USA and the nearest U.S. District Court shall be the exclusive jurisdiction and venue for all legal proceedings that are not arbitrated under this Agreement.

20. **Independent Contractors.** The relationship of the parties is that of independent contractor, and nothing herein shall be construed to create a partnership, joint venture, franchise, employment, or agency relationship between the parties. Licensee shall have no authority to enter into agreements of any kind on behalf of CodeSourcery and shall not have the power or authority to bind or obligate CodeSourcery in any manner to any third party.

21. **Force Majeure.** Neither CodeSourcery nor Licensee shall be liable for damages for any delay or failure of delivery arising out of causes beyond their reasonable control and without their fault or negligence, including, but not limited to, Acts of God, acts of civil or military authority, fires, riots, wars, embargoes, or communications failure.

22. **Miscellaneous.** This Agreement constitutes the entire understanding of the parties with respect to the subject matter of this Agreement and merges all prior communications, representations, and agreements. This Agreement may be modified only by a written agreement signed by the parties. If any provision of this Agreement is held to be unenforceable for any reason, such provision shall be reformed only to the extent necessary to make it enforceable. This Agreement shall be construed under the laws of the State of California, USA, excluding rules regarding conflicts of law. The application of the United Nations Convention of Contracts for the International Sale of Goods is expressly excluded. This license is written in English, and English is its controlling language.