Sourcery G++ Lite

ARM uClinux

Sourcery G++ Lite 2009q3-66

Getting Started

CODESOURCERY

Sourcery G++ Lite: ARM uClinux: Sourcery G++ Lite 2009q3-66: Getting Started

CodeSourcery, Inc.

Copyright © 2005, 2006, 2007, 2008, 2009 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.

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 ARM uClinux"	This chapter contains information about using Sourcery G++ Lite that is specific to ARM uClinux 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 ARM uClinux.
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 ARM uClinux. 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 ARM uClinux is intended for developers working on embedded uClinux applications. It may also be used for uClinux kernel development and debugging, or to build a uClinux 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 ARM uClinux".

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. You will need to identify the multilib appropriate for your target in order to find the correct **gdbserver** executable to use for debugging your applications, as described in Section 3.6, "GDB Server".

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.

Run your program on the target system.   Copy your program to the target system and run it from the command line.

Debug your program on the target using GDB server.   You can debug a program on a remote ARM uClinux target using GDB server. Copy your program to the target system. Follow the instructions in Section 3.6, "GDB Server" to install and run **gdbserver** on your target system. Then, you can connect to GDB server from the debugger running on your host system. Refer to Section 4.3,

---

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

"Running Applications from GDB" for instructions on connecting to the target 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 `arm-uclinuxeabi`.

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 ARM uClinux" 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 Code-Sourcery'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. 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/
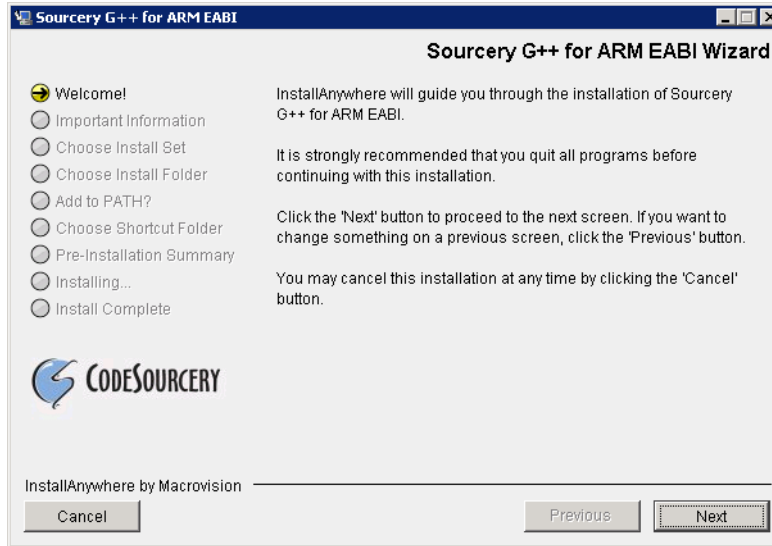
Running the Installer.   The graphical installer guides you through the steps to install Sourcery G++ Lite.

You may want to change the install directory pathname and customize the shortcut installation.
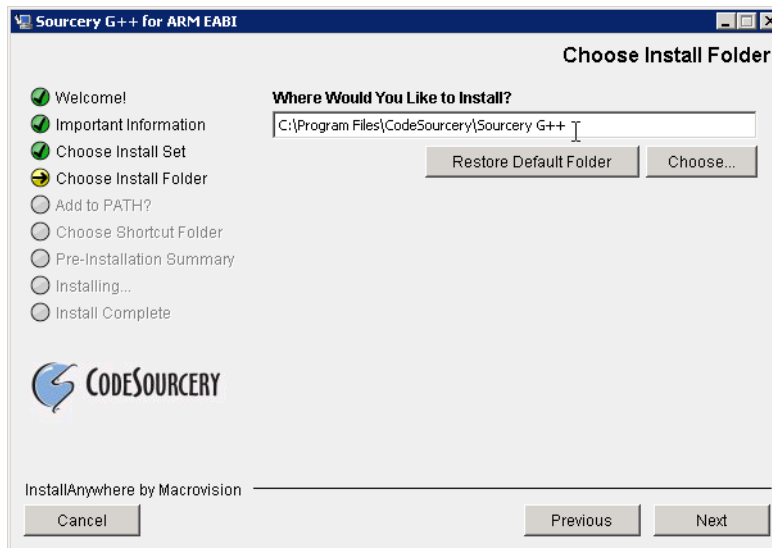
Choose Install Folder.   Select the pathname to your install directory.
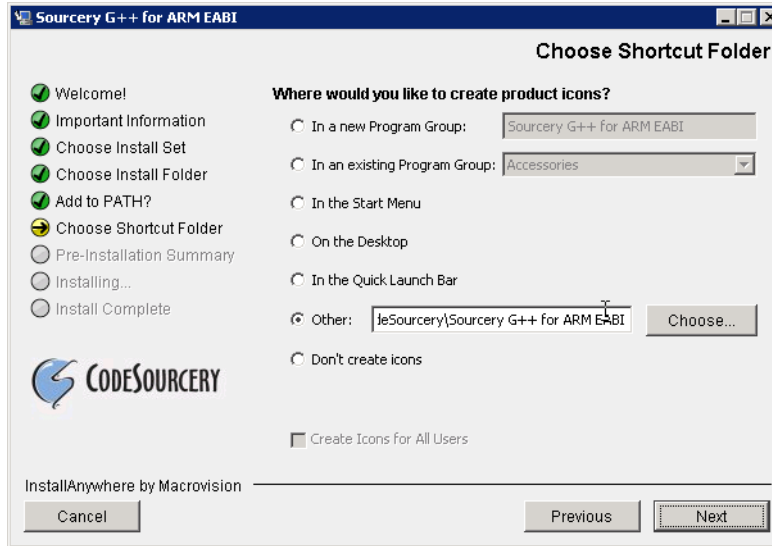
Choose Shortcut Folder.   You can customize where the installer creates
shortcuts for quick access to Sourcery G++ Lite.

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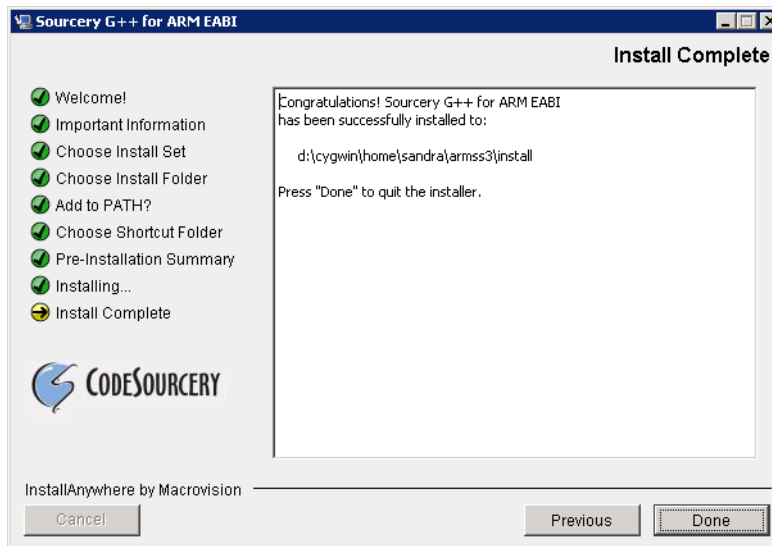
Install Complete.   You should see a screen similar to this after a successful
install.

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 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++-2009q3`.

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 ARM uClinux 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.

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 ARM uClinux target all begin with **arm-uclinuxeabi**. 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:

```
> arm-uclinuxeabi-g++ -v
```

Verify that the last line of the output contains: `Sourcery G++ Lite 2009q3-66`.

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++-arm-uclinuxeabi/man.

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

```
> arm-uclinuxeabi-g++ -v
```

Verify that the last line of the output contains: Sourcery G++ Lite 2009q3-66.

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 ARM uClinux. 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 ARM uClinux. 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 `arm-uclinuxeabi` 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 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 ARM uClinux

This chapter contains information about features of Sourcery G++ Lite that are specific to ARM uClinux 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 ARM uClinux, 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 ARM	2009q3-66	Provided for kernel debugging only. See Chapter 5, "Sourcery G++ Debug Sprite".
GDB Server	N/A	Included with GDB. See Section 3.6, "GDB Server".
Target libraries		
uClibc C Library	0.9.30	
Linux Kernel Headers	2.6.30	
Other utilities		
ELF-to-FLT Conversion Utility	N/A	
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. You can find the sysroot directories provided with Sourcery G++ in the `arm-uclinuxeabi/libc` directory of your installation.

3.2.1. Included Libraries

The following library configurations are available in Sourcery G++ Lite for ARM uClinux.

ARMv4T - Little-Endian, Soft-Float	
Command-line option(s):	default
Sysroot subdirectory:	`./`

ARMv6-M Thumb - Little-Endian, Soft-Float	
Command-line option(s):	`-mthumb -march=armv6-m`
Sysroot subdirectory:	`armv6-m/`

ARMv7 Thumb-2 - Little-Endian, Soft-Float	
Command-line option(s):	`-mthumb -march=armv7 -mfix-cortex-m3-ldrd`
Sysroot subdirectory:	`thumb2/`

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:

```
> arm-uclinuxeabi-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. Using VFP Floating Point

3.3.1. Enabling Hardware Floating Point

GCC provides three basic options for compiling floating-point code:

- Software floating point emulation, which is the default. In this case, the compiler implements floating-point arithmetic by means of library calls.

- VFP hardware floating-point support using the soft-float ABI. This is selected by the `-mfloat-abi=softfp` option. When you select this variant, the compiler generates VFP floating-point instructions, but the resulting code uses the same call and return conventions as code compiled with software floating point.

- VFP hardware floating-point support using the VFP ABI, which is the VFP variant of the Procedure Call Standard for the ARM® Architecture (AAPCS). This ABI uses VFP registers to pass function arguments and return values, resulting in faster floating-point code. To use this variant, compile with `-mfloat-abi=hard`.

You can freely mix code compiled with either of the first two variants in the same program, as they both use the same soft-float ABI. However, code compiled with the VFP ABI is not link-compatible with either of the other two options. If you use the VFP ABI, you must use this option to compile your entire program, and link with libraries that have also been compiled with the VFP ABI. For example, you may need to use the VFP ABI in order to link your program with other code compiled by the ARM RealView® compiler, which uses this ABI.

Sourcery G++ Lite for ARM uClinux includes libraries built with software floating point, which are compatible with VFP code compiled using the soft-float ABI. While the compiler is capable of generating code using the VFP ABI, no compatible runtime libraries are provided for uClinux targets.

Note that, in addition to selecting hard/soft float and the ABI via the `-mfloat-abi` option, you can also compile for a particular FPU using the `-mfpu` option. For example, `-mfpu=neon` selects VFPv3 with NEON coprocessor extensions.

3.3.2. NEON SIMD Code

Sourcery G++ includes support for automatic generation of NEON SIMD vector code. Autovectorization is a compiler optimization in which loops involving normal integer or floating-point code are transformed to use NEON SIMD instructions to process several data elements at once.

To enable generation of NEON vector code, use the command-line options `-ftree-vectorize -mfpu=neon -mfloat-abi=softfp`. The `-mfpu=neon` option also enables generation of VFPv3 scalar floating-point code.

Sourcery G++ also includes support for manual generation of NEON SIMD code using C intrinsic functions. These intrinsics, the same as those supported by the ARM RealView® compiler, are defined in the `arm_neon.h` header and are documented in the 'ARM NEON Intrinsics' section of the GCC manual. The command-line options `-mfpu=neon -mfloat-abi=softfp` must be specified to use these intrinsics; `-ftree-vectorize` is not required.

3.3.3. Half-Precision Floating Point

Sourcery G++ for ARM uClinux includes support for half-precision (16-bit) floating point, including the new `__fp16` data type in C and C++, support for generating conversion instructions when compiling for processors that support them, and library functions for use in other cases.

To use half-precision floating point, you must explicitly enable it via the `-mfp16-format` command-line option to the compiler. For more information about `__fp16` representations and usage from C and C++, refer to the GCC manual.

3.4. ABI Compatibility

The Application Binary Interface (ABI) for the ARM Architecture is a collection of standards, published by ARM Ltd. and other organizations. The ABI makes it possible to combine tools from different vendors, including Sourcery G++ and ARM RealView®.

Sourcery G++ implements the ABI as described in these documents, which are available from the ARM Information Center[1]:

- BSABI - ARM IHI 0036B (10 October 2008)

- BPABI - ARM IHI 0037B (10 October 2008)

- EHABI - ARM IHI 0038A (10 October 2008)

- CLIBABI - ARM IHI 0039A (10 October 2008)

- AADWARF - ARM IHI 0040A (10 October 2008)

---

[1] http://infocenter.arm.com

- CPPABI - ARM IHI 0041B (10 October 2008)

- AAPCS - ARM IHI 0042C (10 October 2008)

- RTABI - ARM IHI 0043B (10 October 2008)

- AAELF - ARM IHI 0044C (10 October 2008)

- ABI Addenda - ARM IHI 0045B (10 October 2008)

Sourcery G++ currently produces DWARF version 2, rather than DWARF version 3 as specified in AADWARF.

3.5. Building uClinux Applications

When you use GCC to link a uClinux application, it creates two output files. The executable file, as specified by the -o command-line option, is a uClinux FLAT format binary (bFLT) file. This is the file you should copy to and run on your uClinux target. The second output file is an ELF-format file containing additional debug and symbol table information to allow you to debug your program with GDB, as described in Section 3.6, "GDB Server". This file has a .gdb extension.

For example, if you specify the command

```
arm-uclinuxeabi-gcc foo.c -o bar
```

then bar is the FLAT-format executable and bar.gdb is the ELF-format file.

3.6. GDB Server

Sourcery G++ Lite contains a **gdbserver** for running on the target. The server executable is located in the *sysroot*/usr/bin directory of your installation, where *sysroot* is the pathname to the sysroot, as documented in Section 3.2, "Library Configurations". You need to copy the appropriate **gdbserver** executable to your target system and then invoke it as

```
# gdbserver :port program
```

port can be any available TCP port; 5000 is a common choice. **gdbserver** waits for a connection from **gdb** and then commences serving requests for it. To connect to **gdbserver** from your host system, start **gdb**, but specify the special .gdb version of your program.

```
> arm-uclinuxeabi-gdb program.gdb
```

Then connect to the target system:

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

At this point you are able to debug as usual.

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 arm-uclinuxeabi, as indicated by the **arm-uclinuxeabi** 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:

```c
#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:

```
> arm-uclinuxeabi-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 **arm-uclinuxeabi-gcc** with **arm-uclinuxeabi-g++**.)

4.2. Running Applications on the Target System

To run your program on a uClinux 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 | arm-uclinuxeabi-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 ARM uClinux target. Follow the instructions in Chapter 3, "Sourcery G++ Lite for ARM uClinux" 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 ARM uClinux.

Sourcery G++ Lite contains the Sourcery G++ Debug Sprite for ARM uClinux. 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 ARM uClinux" 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:

```
> arm-uclinuxeabi-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 ARM Debug Sprite
    (Sourcery G++ Lite Sourcery G++ Lite 2009q3-66)
armusb: [speed=<n:0-7>] ARMUSB device
  armusb:///0B01000C - Stellaris Evaluation Board (0B01000C)
rdi: (rdi-library=<file>&rdi-config=<file>) RDI Device
  rdi:/// - RDI Device
```

This shows that ARMUSB and RDI devices are supported. The exact set of supported devices depends on your host system and the version of Sourcery G++ you have installed; refer to Section 5.2, "Invoking Sourcery G++ Debug Sprite" for complete information.

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:

```
> arm-uclinuxeabi-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 ARM uC-linux:

`rdi`   Use an RDI debugging device. Refer to Section 5.4, "Remote Debug Interface Devices".

`flashpro`   Use a FlashPro debugging device. Refer to Section 5.5, "Actel FlashPro Devices".

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

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

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

5.3. 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 | arm-uclinuxeabi-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. Remote Debug Interface Devices

Remote Debug Interface (RDI) devices are supported. The RDI device URL accepts no hostname, port or path components, so the `device-url` is specified as follows:

```
rdi:[///][?device-options]
```

The following `device-options` are required:

`rdi-library=library`   Specify the library (DLL or shared object) implementing the RDI target you wish to use.

`rdi-config=configfile`   Specify a file containing configuration information for `library`. The format of this file is specific to the RDI library you are using, but tends to constitute a list of `key=value` pairs. Consult the documentation of your RDI library for details.

5.5. Actel FlashPro Devices

On Windows hosts, Sourcery G++ Lite supports FlashPro devices used with Actel Cortex-M1 development kits.

For FlashPro devices, the `device-url` has the following form:

```
flashpro:[//usb12345/][?jtagclock=rate]
```

The optional `usb12345` part indicates the ID of the FlashPro device to connect to, which is useful if you have more than one such device attached to your computer. If the ID is omitted, the Debug Sprite connects automatically to the first detected FlashPro device. You can enumerate the connected FlashPro devices by invoking the Sprite with the `-i` switch, as follows:

```
> arm-uclinuxeabi-sprite -i flashpro:
```

The `jtagclock` option allows the communication speed with the target board to be altered. The `rate` is specified in Hz and may range between 93750 and 4000000. The default is 93750, the slowest speed supported by the FlashPro device. Depending on your target board, you may be able to increase this rate, but beware that communication errors may occur above a certain threshold. If you encounter communication errors with a higher-than-default speed selected, try reducing the speed.

5.5.1. Installing FlashPro Windows drivers

Windows drivers for the FlashPro device are included with the FlashPro software provided by Actel. Refer to Actel's documentation for details on installing this software. You must use the Actel FlashPro software to configure the FPGA on your Cortex-M1 board, but it does not need to be running when using the Debug Sprite.

Once you have set up your board using the FlashPro software, you can check that it is recognized by the Sourcery G++ Debug Sprite by running the following command:

```
> arm-uclinuxeabi-sprite -i
flashpro: [jtagclock=<n:93750-4000000>] FlashPro device
  flashpro://usb12345/ - FlashPro Device
  ...
```

If output similar to the above does not appear, your FlashPro device is not working correctly. Contact CodeSourcery for further guidance in that case.

5.6. 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:

```
> arm-uclinuxeabi-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].

5.7. Supported Board Files

The Sourcery G++ Debug Sprite for ARM uClinux 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
Altera Cyclone III Cortex-M1	`cycloneiii-cm1`
ARMulator (RDI)	`armulator`

---

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

5.8. 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 `arm-uclinuxeabi/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:

banked-regs The `banked-regs` property specifies that the CPU of the target board has banked registers for different processor modes (supervisor, IRQ, etc.).

has-vfp The `has-vfp` property specifies that the CPU of the target board has VFP registers.

system-v6-m The `system-v6-m` property specifies that the CPU of the target board has ARMv6-M architecture system registers.

system-v7-m The `system-v7-m` property specifies that the CPU of the target board has ARMv7-M architecture system registers.

core-family The `core-family` property specifies the ARM family of the target. The body of the `<property>`

element may be one of `arm7`, `arm9`, `arm11`, and `cortex`.

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

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

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

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

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

`<write-memory>` This element writes a value to a memory location. It has three attributes: `address`, `value` and `bits`. The `bits` attribute is optional and defaults to 32. Bit widths of 8, 16 and 32 bits are supported. The address written to must be naturally aligned for the size of the write being done.

`<delay>` This element introduces a delay. It has one attribute, `time`, which specifies the number of milliseconds, or microseconds to delay by.

`<description>` This element encapsulates a human-readable description of its enclosing element.

`<property>` The `<property>` element allows additional name/value pairs to be specified. The property name is specified in a `name` attribute. The property value is the body of the `<property>` element.

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++-arm-uclinuxeabi/` 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++-arm-uclinuxeabi/man/man1
```

Then you can invoke **man** as:

```
> man ./arm-uclinuxeabi-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 ARM uClinux. You should read through these notes to learn about new features and bug fixes.

A.1. Changes in Sourcery G++ Lite for ARM uClinux

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

A.1.1. Changes in Sourcery G++ Lite 2009q3-66

GDB crash fix. A GDB bug has been fixed that caused GDB to crash when unloading shared libraries or switching executables.

@`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.

NEON improvements. The compiler now generates improved NEON vector code when copying memory or storing constants to memory using the NEON coprocessor. The compiler also generates better code for accessing data arrays that are not known to have 64-bit alignment. In addition, a bug that caused internal compiler errors when compiling for Thumb-2 with NEON enabled has been fixed, as has another bug that caused some vector shift NEON operations to be wrongly rejected.

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.

Compiler errors with `float32_t`. A bug has been fixed that caused compiler errors when using the `float32_t` type from `arm_neon.h`.

Support for ARM Cortex-A5 cores. Sourcery G++ now includes basic support for ARM Cortex-A5 cores. Use the `-mcpu=cortex-a5` command-line option.

Static variables and `asm` statements bug fix. A bug in GCC that caused functions containing static variables and `asm` statements to be miscompiled at `-O2` or above has been fixed. The bug also occurred at other optimization levels when the `-fremove-local-statics` command-line option was used.

Warnings for naked functions. A compiler bug that resulted in incorrect warnings about missing return statements in non-void functions declared with the `naked` attribute has been fixed.

Optimizer bug fix. A bug in GCC that caused functions with complex loop nests to be miscompiled at `-O2` or above has been fixed. The bug also occurred at other optimization levels when the `-fpromote-loop-indices` command-line option was used.

VFPv4 support. Sourcery G++ now includes support for VFPv4, VFPv4-D16 and NEON-VFPv4 coprocessors. Use the `-mfpu=vfpv4`, `-mfpu=vfpv4-d16` or `-mfpu=neon-vfpv4` options, respectively.

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 2009q3-36

Improved optimization for ARM.   GCC now automatically enables loop unrolling and `-fpromote-loop-indices` when `-O2` or `-O3` is specified. Loop unrolling is limited at `-O2` to control code growth. These changes improve performance by more than 5%.

VFP assembly mnemonics.   The assembler now accepts unified assembly mnemonics for VFP instructions (e.g. `VADD.f32 s0, s0`) in legacy syntax mode.

ARM Cortex-R4F assembler bug fix.   The assembler now correctly recognizes the `-mcpu=cortex-r4f` command-line option to select the Cortex-R4F processor.

VFP half-precision extensions.   Sourcery G++ now includes support for VFP coprocessors with half-precision floating-point extensions. This can be enabled with the `-mfpu=vfpv3-d16-fp16` or `-mfpu=vfpv3-fp16` command-line options.

Linux kernel headers update.   Linux kernel header files have been updated to version 2.6.30.

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.

Improved optimization for Thumb-2.   GCC now supports instruction scheduling for Thumb-2 code. This optimization is enabled when compiling with `-O2`, `-O3`, or `-Os`, and can improve performance substantially.

ARM VFP assembler bug fix.   The assembler now correctly assembles the `vmls`, `vnmla` and `vnmls` mnemonics. Previously these were incorrectly assembled to different instructions.

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.

Mixed PIC and non-PIC code.   The **elf2flt** utility, automatically run by Sourcery G++ when linking uClinux applications, now gives an error when an attempt is made to link mixed PIC and non-PIC code, which is not supported on uClinux targets. Formerly, it silently produced an invalid executable in such cases.

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.

New assembler directive `.inst`.   The assembler now accepts the new `.inst` directive to generate an instruction from its integer encoding.

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.

Internal error in assembler. An assembler bug that caused an internal error when `.thumb` or `.arm` appears after an invalid instruction has been fixed.

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.).`

Improved bit counting operation. The `__builtin_ctz` built-in function, which returns the number of trailing zero bits in a value, has been improved to use a shorter instruction sequence for ARMv6T2 and later.

Out-of-range branch errors. A Thumb-2 code generation defect in the compiler that caused `branch out of range` errors from the assembler has been eliminated.

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

Assembler validation improvements. The assembler now issues a warning when a section finishes with an unclosed IT instruction block at the end of the input file. It also now rejects unwinding directives that appear outside of a `.fnstart`/`.fnend` pair. Additionally, 32-bit Thumb instructions are now correctly rejected when assembling for cores that do not support these instructions.

Assembler validations fix. A bug in the assembler that caused some `addw` and `subw` instructions with SP or PC as operand to be wrongly rejected has been fixed.

-mauto-it assembler option replaced with -mimplicit-it. The `-mauto-it` command-line option to the assembler has been replaced with a more general `-mimplicit-it` option to control the behavior of the assembler when conditional instructions appear outside an IT instruction block. If you were previously using `-mauto-it`, you should now use `-mimplicit-it=always`. Other `-mimplicit-it` modes allow you to separately control implicit IT instruction insertion behavior in ARM and Thumb-2 code. For more information, refer to the assembler manual. In addition to renaming the option, a number of bugs in the implicit IT generation have been fixed.

GDB backwards compatibility fix. A bug has been fixed that caused GDB to crash when loading symbols from binaries built by very old versions of GCC.

Debug information for variadic functions. A compiler bug that resulted in incorrect debug information for functions with variable arguments has been fixed.

Overlay sections. **arm-uclinuxeabi-readelf** now correctly recognizes section headers for `ARM_DEBUGOVERLAY` and `ARM_OVERLAYSECTION` sections.

Code generation improvements. The compiler has been changed to make better use of VFP registers in mixed integer and floating-point code, resulting in faster code.

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.`

Assembler fix for mixed Thumb and ARM mode. A bug in the assembler has been fixed where mapping symbols were sometimes incorrectly placed at section boundaries. This could lead to incorrect disassembly in some cases.

C++ exception matching.   A C++ conformance defect has been fixed. According to clause 15.3 of the standard, given a derived class `D` with base `B`, a thrown `D *` object is not caught by a handler with type `B *&` (that is, a reference to pointer `B`). The compiler formerly treated this case incorrectly as if the handler had type `B *`, which does catch `D *`.

`-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];`).

Linker bug fix.   A bug that caused the linker to crash when `.ARM.exidx` sections were discarded by a linker script has been fixed.

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.

uClibc upgrade.   uClibc has been updated to version 0.9.30 plus additional updates from the uclibc.org repository as of June 2009. Programs linked with uClibc shared libraries from a previous version of Sourcery G++ must be recompiled to run with the new version of uClibc.

GCC version 4.4.1.   Sourcery G++ Lite for ARM uClinux 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`.

Watchpoint support.   The Sourcery G++ Debug Sprite now implements watchpoints on all currently-supported debugging devices.

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 2009q1-163

Incorrect placement of linker-generated functions.   A bug that caused some linker-generated functions (including stubs to support interworking from ARM mode to Thumb mode and stubs to avoid processor errata) to be placed in data sections has been fixed.

New option for automatically generating IT blocks.   The assembler now allows use of conditional Thumb-2 instructions without requiring explicit IT instructions. Use the `-mauto-it` command-line option to enable this automatic generation of IT instructions.

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.

Assembler bug fix. A bug in the assembler that caused duplicate and missing mapping symbols has been fixed. The bug caused incorrect **objdump** output and incorrect byte-swapping for BE8 configurations.

Stack backtracing and C++ exception handling. Improvements have been made to the linker in support of C++ runtime exception handling and stack backtracing. A problem that caused crashes during the backtrace of C routines that were not compiled with the `-fexceptions` option has been fixed. In addition, the linker generates more compact stack unwinding tables which can lead to smaller executables.

Incorrect linker-generated functions. A bug that caused some linker-generated functions (such as stubs to support interworking from ARM mode to Thumb mode) to contain only `nop` instructions instead of correct code sequences has been fixed.

Assembler diagnostics for invalid instructions. The assembler now issues diagnostics for invalid `ADR` and `ADRL` instructions. Formerly, these invalid instructions were silently mis-assembled. This assembler bug did not affect correct code.

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.

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.

Disassembler bug fix. A bug has been fixed that caused incorrect disassembly of some object files with multiple sections whose symbol tables included symbols in the middle of functions. These typically resulted from hand-written assembly.

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.

Linker crash with very large applications. A linker bug that caused a crash when linking very large applications with the `--fix-cortex-a8` command-line option has been fixed.

Assembler marking of data. Data generated using the assembler directives `.ascii`, `.asciz`, `.dc.d`, `.dc.s`, `.dc.x`, `.dcb`, `.dcb.b`, `.dcb.d`, `.dcb.l`, `.dcb.s`, `.dcb.w`, `.dcb.x`, `.ds`, `.ds.b`, `.ds.d`, `.ds.l`, `.ds.p`, `.ds.s`, `.ds.w`, `.ds.x`, `.double`, `.fill`, `.float`, `.incbin`, `.single`, `.space`, `.skip`, `.string`, `.string8`, `.string16`, `.string32`, `.string64`, and `.zero` is now correctly marked by the assembler as data rather than code. This fixes incorrect byte-swapping of such data when linking for BE8 configurations.

arm-uclinuxeabi-objcopy bug fix. A bug has been fixed that caused **arm-uclinuxeabi-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.

Cortex-A8 erratum workaround enabled for ARMv7-A. The workaround for the erratum in Cortex-A8 processors mentioned below is now enabled by default if you are targeting the ARMv7-A architecture profile. The workaround can be disabled by passing the `--no-fix-cortex-a8` option to the linker.

Improved vectorization. Automatic vectorization for NEON now uses the fused multiply-add (`VMLA`) and fused multiply-subtract (`VMLS`) instructions. These fused instructions are faster than the equivalent two-instruction sequence consisting of a multiply followed by an add or subtract.

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

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.

Out-of-bounds accesses to stack arrays. A bug has been fixed that caused internal compiler errors when some code involving out-of-bounds accesses to stack-allocated arrays was compiled with the `-mthumb` option. Such code is not valid C; although it is now accepted by the compiler and no diagnostic is issued, it has undefined behavior if executed.

Erratum workaround for Cortex-A8 processors. The linker now implements a workaround for an erratum in Cortex-A8 processors. If you are targeting an affected part and wish to use the workaround, pass the `--fix-cortex-a8` option to the linker. Please contact ARM for further details of the erratum.

Maximum code alignment increased. The maximum allowed code alignment has been increased from 32 to 64 bytes. This change affects the `.p2align` and `.align` directives in GAS and the `-falign-functions` GCC option.

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.

A.1.4. Changes in Sourcery G++ Lite 2009q1-118

GCC version 4.3.3. Sourcery G++ Lite for ARM uClinux 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`.

Improved NOP generation for Thumb-2 cores. The assembler now generates Thumb-2/ARMv6K architectural NOP instructions when alignment padding is required in code sections.

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`.

CS3 board and processor support. CS3 board and processor support has been cleaned up to remove entries that are not appropriate for or supported by Sourcery G++ Lite on ARM uClinux targets. This includes processors for which Sourcery G++ Lite does not include appropriate run-time libraries. In addition, CS3 support for boards based on Cortex-M3 processors has been removed as these boards are not sufficiently powerful to run uClinux. These changes are intended to simplify processor and board selection.

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.

Linker errors on non-ELF input.   A bug has been fixed that caused internal errors from the linker when linking non-ELF input files (with the `-b` or `--format` linker options).

Undefined weak references in shared libraries.   A linker bug has been fixed affecting calls from Thumb code in shared libraries to functions that are undefined weak references when the shared library is linked. Such calls executed as nops whether or not the functions were defined at run time.

uClibc `splice`, `vmsplice` and `tee` functions.   uClibc now provides the functions `splice`, `vmsplice` and `tee`.

Improved code generation.   The compiler has been improved to generate better code for an integer multiplication whose result feeds into an addition.

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.

Performance improvements.   Tuning parameters for ARM code generation have been adjusted to improve performance of the generated code.

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.

`CMP` Thumb-2 instruction.   The assembler no longer issues an error about `CMP` instructions in which the second argument is the stack pointer (`r13`), as these are valid instructions. However, use of the stack pointer in this context is deprecated in the current ARM architecture specification and the assembler now warns about the deprecated use.

`DMB`, `DSB`, and `ISB` instructions on ARMv6-M.   The assembler now accepts the `DMB`, `DSB`, and `ISB` instructions on ARMv6-M CPUs, including Cortex-M0 and Cortex-M1. These instructions were incorrectly rejected on these CPUs in previous releases.

Thumb half-precision floating point bug fix.   A compiler bug has been fixed that formerly caused incorrect code to be generated in Thumb mode for functions using half-precision floating-point constants. The bug did not affect Thumb-2 code.

Improved code generation.   The compiler has been improved to generate better code for integer multiplication by certain constants.

Thumb-2 `switch` code generation bug fix.   A bug has been fixed that caused incorrect Thumb-2 code to be generated for some `switch` statements.

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

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.

Internal compiler error with large NEON types.   A bug has been fixed that caused internal compiler errors when compiling code using NEON types at least 32 bytes wide.

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.

Vectorized shift fix.   A bug has been fixed that caused incorrect code for loops containing a right shift by a constant. The bug affected code compiled with `-mfpu=neon` and loop vectorization enabled with `-O3` or `-ftree-vectorize`.

Incorrect code for nested functions.   A bug in GCC that caused the compiler to generate incorrect code for nested functions has been fixed. The bug resulted in incorrect stack alignments in the affected functions.

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

ARM build attributes conformance improvements.   Several ARM EABI 2.07 conformance issues relating to the handling of build attributes in the assembler and linker have been fixed. All build attribute types are now recognized, and can now be declared by name, in addition to by number. Support for merging attributes in the linker has been improved, and the linking of incompatible objects is now detected and rejected in more cases.

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.

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

Linker crash on incompatible input files.   Some third-party compilers, including ARM RealView® 4.0, produce a build attribute marking output files that are not compatible with the ABI for the ARM Architecture. This attribute sometimes caused the linker to crash. The linker now correctly issues an error message.

A.1.5. Changes in Sourcery G++ Lite 2008q3-68

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.

Optimizer bug fix.   A bug that caused an `unrecognizable insn` internal compiler error when compiling at optimization levels above `-O0` has been fixed.

VFP compiler fix.   A compiler bug that resulted in `internal compiler error: output_ operand: invalid expression as operand` when generating VFP code has been fixed.

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.

Workaround for Cortex-M3 CPU errata. Errata present in some Cortex-M3 cores can cause data corruption when overlapping registers are used in `LDRD` instructions. The compiler avoids generating these problematic instructions when the `-mfix-cortex-m3-ldrd` or `-mcpu=cortex-m3` command-line options are used. The Sourcery G++ runtime libraries have also been updated to include this workaround.

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.

Misaligned NEON memory accesses. A bug has been fixed that caused the compiler to use aligned NEON load/store instructions to access misaligned data when autovectorizing certain loops. The bug affected code compiled with `-mfpu=neon` and loop vectorization enabled with `-O3` or `-ftree-vectorize`.

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.

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.

A.1.6. Changes in Sourcery G++ Lite 2008q3-42

Definition of `va_list`. In order to conform to the ABI for the ARM Architecture, the definition of the type of va_list (defined in `stdarg.h`) has been changed. This change impacts only the mangled names of C++ entities. For example, the mangled name of a C++ function taking an argument of type `va_list`, or `va_list *`, or another type involving `va_list` has changed. Since this is an incompatible change, you must recompile and relink any modules defining or using affected `va_list`-typed entities.

Thumb-2 assembler fixes. The Thumb-2 encodings of `QADD`, `QDADD`, `QSUB`, and `QDSUB` have been corrected. Previous versions of the assembler generated incorrect object files for these instructions. The assembler now accepts the `ORN`, `QASX`, `QSAX`, `RRX`, `SHASX`, `SHSAX`, `SSAX`, `USAX`, `UHASX`, `UQSAX`, and `USAX` mnemonics. The assembler now detects and issues errors for invalid uses of register 13 (the stack pointer) and register 15 (the program counter) in many instructions.

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**.

Improved support for debugging RealView® objects . GDB support for programs compiled by the ARM RealView® compiler has been improved.

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.

NEON improvements. Several improvements and bug fixes have been made to the NEON Advanced SIMD Extension support in GCC. A problem that caused the autovectorizer to fail in some

circumstances has been fixed. Also, many of the intrinsics available via the `arm_neon.h` header file now have improved error checking for out-of-bounds arguments, and the `vget_lane` intrinsics that return signed values now produce improved code.

NEON compiler fix.   A compiler bug that resulted in incorrect NEON code being generated has been fixed. Typically the incorrect code occurred when NEON intrinsics were used inside small `if` statements.

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`.

Mixed-case NEON register aliases.   An assembler bug that prevented NEON register aliases from being created with mixed-case names using the `.dn` and `.qn` directives has been fixed. Previously only aliases created with all-lowercase or all-uppercase names worked correctly.

Improvements to elf2flt utility.   The **elf2flt** utility, automatically run by Sourcery G++ when linking uClinux applications, is now compatible with the linker option `--gc-sections`. Previously, applications linked with `--gc-sections` terminated at startup with an illegal instruction error.

Janus 2CC support.   GCC now includes a work-around for a hardware bug in Avalent Janus 2CC cores. To compile and link for these cores, use the `-mfix-janus-2cc` compiler option. If you are using the linker directly use the `--fix-janus-2cc` linker option.

ARM exception handling bug fix.   A bug in the runtime library has been fixed that formerly caused throwing an unexpected exception in C++ to crash instead of calling the unexpected exception handler. The bug only affected C++ code compiled by non-GNU compilers such as ARM RealView®.

Mangling of NEON type names.   A bug in the algorithm used by the C++ compiler for mangling the names of NEON types, such as `int8x16_t`, has been fixed. These mangled names are used internally in object files to encode type information in addition to the programmer-visible names of the C++ variables and functions. The new mangled name encoding is more compact and conforms to the ARM C++ ABI.

LinuxThreads support.   The included uClibc now supports the LinuxThreads implementation of POSIX threads for the ARMv4T multilib. Please note that this feature is not yet supported by the ARMv6-M Thumb and ARMv7 Thumb-2 multilibs.

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.

Half-precision floating point.   Sourcery G++ now includes support for half-precision floating point via the `__fp16` type in C and C++. The compiler can generate code using either hardware support or library routines. For more information, see Section 3.3.3, "Half-Precision Floating Point".

A.1.7. Changes in Sourcery G++ Lite 2008q3-9

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, improved Thumb mode support, 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.

Uppercase operands to IT instructions.   The assembler now accepts both uppercase and lowercase operands for the `IT` family of instructions.

NEON autovectorizer fix.   A compiler bug that caused generation of bad `VLD1` instructions has been fixed. The bug affected code compiled with `-mfpu=neon -ftree-vectorize`.

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.

Memory access errors when setting breakpoints.   A GDB bug that caused spurious "Cannot access memory" errors has been fixed. The errors occurred when setting breakpoints after the program being debugged exited or was killed.

ARMv7 offset out of range errors.   An assembler bug that resulted in `offset out of range` errors when compiling for ARMv7 processors has been fixed.

Thumb-2 `MUL` encoding.   In Thumb-2 mode, the assembler now encodes `MUL` as a 16-bit instruction (rather than as a 32-bit instruction) when possible. This fix results in smaller code, with no loss of performance.

ARM C++ ABI utility functions.   Vector utility functions required by the ARM C++ ABI no longer crash when passed null pointers. The affected functions are `__aeabi_vec_dtor_cookie`, `__aeabi_vec_delete`, `__aeabi_vec_delete3`, and `__aeabi_vec_delete3_nodtor`. These functions are not intended for use by application programmers; they are only called by compiler-generated code. They are not presently used by the GNU C++ compiler, but are used by some other compilers, including ARM's RealView® compiler.

GCC version 4.3.2.   Sourcery G++ Lite for ARM uClinux 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`.

Smaller Thumb-2 code.   When optimizing for size (i.e., when `-Os` is in use), GCC now generates the 16-bit `MULS` Thumb-2 multiply instruction instead of the 32-bit `MUL` instruction.

Thumb-2 `RBIT` encoding.   An assembler bug that resulted in incorrect encoding of the Thumb-2 `RBIT` instruction has been fixed.

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.

Marvell Feroceon compiler bug fix.   A bug that caused an internal compiler error when optimizing for Marvell Feroceon CPUs has been fixed.

Misaligned accesses to packed structures fix.   A bug that caused GCC to generate misaligned accesses to packed structures has been fixed.

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.1.8. Changes in Older Releases

For information about changes in older releases of Sourcery G++ Lite for ARM uClinux, 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 ARM uClinux 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 ARM	CodeSourcery License
uClibc C Library	GNU Lesser General Public License 2.1 [4]
Linux Kernel Headers	GNU General Public License 2.0 [5]
ELF-to-FLT Conversion Utility	GNU General Public License 2.0 [6]
GNU Make	GNU General Public License 2.0 [7]
GNU Core Utilities	GNU General Public License 2.0 [8]

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++-arm-uclinuxeabi-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
[8] 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.

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, INCID-

ENTAL, 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 WHAT-SOEVER, 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.