Sourcery G++ Lite

ARM GNU/Linux

Sourcery G++ Lite 2009q1-203

Getting Started

CodeSourcery

Sourcery G++ Lite: ARM GNU/Linux: Sourcery G++ Lite 2009q1-203: Getting Started

CodeSourcery, Inc.

Abstract

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

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

Table of Contents

Preface

This preface introduces *Getting Started With Sourcery G++ Lite*. It explains the structure of this guide and lists other sources of information that relate to Sourcery G++ Lite.

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, *Sourcery G++ Lite Licenses*	This chapter provides information about the software licenses that apply to Sourcery G++ Lite. Read this chapter to understand your legal rights and obligations as a user of Sourcery G++ Lite.
Chapter 2, *Sourcery G++ Subscriptions*	This chapter provides information about Sourcery G++ subscriptions. CodeSourcery customers with Sourcery G++ subscriptions receive comprehensive support for Sourcery G++. Read this chapter to find out how to obtain and use a Sourcery G++ subscription.
Chapter 3, *Sourcery G++ Lite for ARM GNU/Linux*	This chapter provides information about this release of Sourcery G++ Lite including any special installation instructions, recent improvements, or other similar information. You should read this chapter before building applications with Sourcery G++ Lite.
Chapter 4, *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 5, *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 6, *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 GNU/Linux.
Chapter 7, *Next Steps with Sourcery G++*	This chapter describes where you can find additional documentation and information about using Sourcery G++ Lite and its components.

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
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 chapter explains what licenses apply to your use of Sourcery G++ Lite. You should read this chapter to understand your legal rights and obligations as a user of Sourcery G++ Lite.

1.1. Licenses for Sourcery G++ Lite Components

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

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

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

The CodeSourcery License is available in Section 1.2, “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.

1.2. 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.**

[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

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.

4. **License Grant to Proprietary Components of the Software.** You are granted a non-exclusive, royalty-free license to install and use the CodeSourcery Proprietary Components of the Software, transmit the CodeSourcery Proprietary Components over an internal computer network, and/or copy the CodeSourcery Proprietary Components for Your internal use only.

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; 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, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND, INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER MALFUNCTION, OR ANY OTHER KIND OF COMMERCIAL DAMAGE, EVEN IF CODESOURCERY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY TO THE EXTENT PROHIBITED BY APPLICABLE LAW. IN NO EVENT SHALL CODESOURCERY'S LIABILITY FOR ACTUAL DAMAGES FOR ANY CAUSE WHATSOEVER, AND REGARDLESS OF THE FORM OF ACTION, EXCEED THE AMOUNT PAID BY YOU IN FEES UNDER THIS AGREEMENT DURING THE PREVIOUS ONE YEAR PERIOD.

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

By downloading or using the Software, Licensee agrees to the foregoing and represents and warrants that it complies with these conditions.

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

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

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

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

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

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

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

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

Chapter 2 Sourcery G++ Subscriptions

CodeSourcery provides support contracts for Sourcery G++. This chapter describes these contracts and explains how CodeSourcery customers can access their support accounts.

2.1. About Sourcery G++ Subscriptions

CodeSourcery offers Sourcery G++ subscriptions. Professional Edition subscriptions provide unlimited support, with no per-incident fees. CodeSourcery's support 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++ to resolve critical problems. Personal Edition subscriptions do not include support, but do include free upgrades as long as the subscription remains active.

CodeSourcery's support is provided by the same engineers who build Sourcery G++. A Sourcery G++ subscription is like having a team of compiler engineers and programming language experts available as consultants!

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. And the board initialization performed by each Sprite can be customized with simple XML-based configuration files to insert delays and write to particular memory addresses. Debug Sprites included in Lite editions of Sourcery G++ include only a subset of the functionality of the Sprites in the subscription editions.
- **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**.
- **CS3.** CS3 provides a uniform, cross-platform approach to board initialization and interrupt handling on ARM EABI, ColdFire ELF, fido ELF, and Stellaris EABI platforms.
- **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.

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>`.

2.2. Accessing your Sourcery G++ Subscription Account

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>`.

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

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

This chapter contains information about using Sourcery G++ Lite on your target system. This chapter also contains information about changes in this release of Sourcery G++ Lite. You should read this chapter to learn how to best use Sourcery G++ Lite on your target system.

3.1. Library Configurations

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

Each multilib corresponds to a *sysroot* directory that contains the files that should be installed on the target system. The sysroot contains the dynamic linker used to run your applications on the target as well as the libraries. Refer to Section 3.2.7, “Using Sourcery G++ Lite on GNU/Linux Targets” for instructions on how to install and use these support files on your target GNU/Linux system. You can find the sysroot directories provided with Sourcery G++ in the `arm-none-linux-gnueabi/libc` directory of your installation. In the tables below, the dynamic linker pathname is given relative to the corresponding sysroot.

3.1.1. Included Libraries

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

ARMv5T - Little-Endian, Soft-Float, GLIBC	
Command-line option(s):	default
Sysroot subdirectory:	`./`
Dynamic linker:	`lib/ld-linux.so.3`

ARMv4T - Little-Endian, Soft-Float, GLIBC	
Command-line option(s):	`-march=armv4t`
Sysroot subdirectory:	`armv4t/`
Dynamic linker:	`lib/ld-linux.so.3`

ARMv7-A Thumb-2 - Little-Endian, Soft-Float, GLIBC	
Command-line option(s):	`-mthumb -march=armv7-a`
Sysroot subdirectory:	`thumb2/`
Dynamic linker:	`lib/ld-linux.so.3`

3.1.2. 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-none-linux-gnueabi-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.

3.2. Using Sourcery G++ Lite for ARM GNU/Linux

3.2.1. Target Kernel Requirements

The GNU C library supplied with Sourcery G++ Lite uses the EABI-based kernel syscall interface. This means applications compiled with Sourcery G++ require at least a 2.6.16 kernel with EABI syscalls enabled.

To provide VFP and Advanced SIMD registers, **gdbserver** requires support from the Linux kernel. Linux 2.6.30 includes the necessary support; for older versions, visit the Sourcery G++ Knowledge Base[1].

3.2.2. Compiling for ARMv4T Systems

By default Sourcery G++ generates Linux binaries that require an ARMv5 or later CPU. To build applications or libraries capable of running on ARMv4T CPUs, use the `-march=armv4t` command-line option. This option also selects libraries for ARMv4T processors; see Section 3.1, “Library Configurations” for details.

Code compiled for ARMv4T is ABI compatible with ARMv5 code. Code and binaries compiled for different architectures may be mixed freely.

3.2.3. GLIBC Backtrace Support

Sourcery G++ supports the `backtrace` function from GLIBC. Backtracing is supported regardless of optimization, with or without a frame pointer, and in both ARM and Thumb modes.

In order to support backtracing, Sourcery G++ enables generation of unwind tables by default when compiling. These tables are used for any stack traversal, including `backtrace`, C++ exception handling, and POSIX thread cancellation. Where none of these are required, you can reduce application size by compiling with `-fno-unwind-tables`.

Some stand-alone programs, including bootloaders and the Linux kernel, can not be built with unwind tables. To accomodate these programs, Sourcery G++ does not generate unwind tables for C code if the `-mabi` or `-ffreestanding` options are used. These options are not generally used in user-space programs. If you use either of these options and do need unwind tables, specify `-funwind-tables` on the **arm-none-linux-gnueabi-gcc** command line.

3.2.4. Using VFP Floating Point

3.2.4.1. 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.

[1] http://support.codesourcery.com/GNUToolchain/kbentry117

- 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 GNU/Linux 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 in Sourcery G++ Lite. However, VFP hard-float libraries built with both ABIs are available to Sourcery G++ Professional Edition subscribers.

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.

3.2.4.2. 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.

3.2.4.3. Half-Precision Floating Point

Sourcery G++ for ARM GNU/Linux 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.

3.2.4.3.1. Representations

ARM supports two incompatible representations for half-precision floating-point values. You must choose one of the representations and use it consistently in your program. The linker gives an error if objects compiled or assembled with different half-precision float attributes are combined in the same executable.

Compiling or assembling with `-mfp16-format=ieee` selects the representation defined in the *IEEE 754-2008* standard, with 1 sign bit, 5 exponent bits, and 10 significand bits (11 bits of significand precision, approximately 3 decimal digits). This format is capable of representing normalized values in the range of 2^{-14} to 65504. It includes support for infinities and NaNs, following the usual IEEE 754 rules.

ARM also supports an alternative half-precision representation, which you can select with `-mfp16-format=alternative`. This format does not include support for infinities and NaNs. Instead, the range of exponent values is extended, so that this format can represent normalized values in the range of 2^{-14} to 131008.

The default for this option is `-mfp16-format=none`, which disables support for half-precision floats.

3.2.4.3.2. C and C++ Usage

When you compile with `-mfp16-format=ieee` or `-mfp16-format=alternative`, GCC defines the `__fp16` data type to represent half-precision float values. Objects of this type have a size of 2 bytes and a natural alignment of 2 bytes.

The `__fp16` type is a storage format only. For purposes of arithmetic and other operations, `__fp16` values are automatically promoted to `float`. In addition, you cannot declare a function with a return value or parameters of type `__fp16`.

Note that conversions from `double` to `__fp16` involve an intermediate conversion to `float`. Because of rounding, this can sometimes produce a different result than a direct conversion.

3.2.4.3.3. Hardware and Library Support

ARM provides hardware support for conversions between `__fp16` and `float` values as an extension to VFP and NEON (Advanced SIMD). GCC generates code using the instructions provided by this extension if you compile with the options `-mfpu=neon-fp16 -mfloat-abi=softfp`, in addition to the `-mfp16-format` option to select a half-precision format.

In other cases, conversions between `__fp16` and `float` values are implemented as library calls.

3.2.5. 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[2]:

- 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)

[2] 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.

3.2.6. Object File Portability

It is possible to create object files using Sourcery G++ for ARM EABI that are link-compatible with the GNU C library provided with Sourcery G++ for ARM GNU/Linux as well as with the Newlib C library provided with the ARM EABI toolchain. These object files are additionally link-compatible with other ARM C Library ABI-compliant static linking environments and toolchains.

To use this feature, when compiling your files with the bare-metal ARM EABI toolchain define the preprocessor constant `_AEABI_PORTABILITY_LEVEL` to 1 before including any system header files. For example, pass the option `-D_AEABI_PORTABILITY_LEVEL=1` on your compilation command line. No special options are required when linking the resulting object files. When building applications for ARM EABI, files compiled with this definition may be linked freely with those compiled without it.

Files compiled in this manner may not use the functions `fgetpos` or `fsetpos`, or reference the type `fpos_t`. This is because Newlib assumes a representation for `fpos_t` that is not AEABI-compliant.

Note that object files are only portable from EABI to GNU/Linux, and not vice versa; object files compiled for ARM GNU/Linux targets cannot be linked into ARM EABI executables.

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

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

The runtime support files, referred to as the *sysroot*, are found in the `arm-none-linux-gnueabi/libc` directory of your Sourcery G++ Lite installation. The sysroot consists of the contents of the `etc`, `lib`, `sbin`, and `usr` directories. There may be other directories in `arm-none-linux-gnueabi/libc` that contain additional sysroots customized for particular combinations of command-line compiler flags, or *multilibs*. Refer to Section 3.1, "Library Configurations" for a list of the included multilibs in this version of Sourcery G++ Lite.

Note for Windows Host Users

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

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

- You can install the files in the filesystem root, replacing the system-provided files. All applications automatically use the Sourcery G++ libraries. This method is primarily useful when you are building a GNU/Linux system from scratch. Otherwise, overwriting your existing C library may break other applications on your system, or cause it to fail to boot.
- You can install the sysroot in an alternate location and build your application with the `-rpath` and `--dynamic-linker` linker options to specify the sysroot location.
- You can install the sysroot in an alternate location and explicitly invoke your application through the dynamic linker to specify the sysroot location. If you are just getting started with Sourcery G++ Lite, this may be the easiest way to get your application running, but this method does not support use of the debugger.

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

3.2.7.1. Installing the Sysroot

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

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

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

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

You need to install the sysroot(s) corresponding to the compiler options you are using for your applications. The tables in Section 3.1, “Library Configurations” tell you which sysroot directories correspond to which compiler options. If you are unsure what sysroot is being referenced when you build your program, you can identify the sysroot by adding `-v` to your compiler command-line options, and looking at the `--sysroot=` pathname in the compiler output.

3.2.7.2. Using Linker Options to Specify the Sysroot Location

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

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

1. First find the correct sysroot directory, dynamic linker, and library subdirectory for your selected multilib. Refer to Section 3.1, “Library Configurations”. In the following steps, *`sysroot`* is the absolute path to the sysroot directory on the target corresponding to your selected multilib. For the default multilib, the dynamic linker path relative to the sysroot is `lib/ld-linux.so.3`, and the library subdirectory is `lib`. This is used in the example below.

2. When invoking **arm-none-linux-gnueabi-gcc** to link your executable, include the command-line options:

   ```
   -Wl,-rpath=sysroot/lib:sysroot/usr/lib \
   -Wl,--dynamic-linker=sysroot/lib/ld-linux.so.3
   ```

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

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

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

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

or

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

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

3.2.7.3. Specifying the Sysroot Location at Runtime

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

To do this, follow these steps:

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

2. Find the correct sysroot directory, dynamic linker, and library subdirectory for your selected multilib. Refer to Section 3.1, “Library Configurations”. In the following steps, *`sysroot`* is the absolute path to the sysroot directory on the target corresponding to your selected multilib. For the default multilib, the dynamic linker is `lib/ld-linux.so.3`, and the library subdirectory is `lib`. This is used in the example below.

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

   ```
   > sysroot/lib/ld-linux.so.3 \
     --library-path sysroot/lib:sysroot/usr/lib \
     /path/to/your-executable
   ```

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

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

3.2.8. Using GDB Server for Debugging

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

3.2.8.1. Running GDB Server

The GDB server executables are included in the sysroot in ABI-specific subdirectories of *`sysroot`*`/usr`. Use the executable from the sysroot and library subdirectory that match your program. See Section 3.1, “Library Configurations” for details.

You must copy the sysroot to your target system as described in Section 3.2.7.1, “Installing the Sysroot”. You must also copy the executable you want to debug to your target system.

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

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

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

If you have installed the sysroot in an alternate directory, invoking **gdbserver** becomes more complicated. You must build your application using the link-time options to specify the location of the sysroot, as described in Section 3.2.7.2, “Using Linker Options to Specify the Sysroot Location”. You must also invoke **gdbserver** itself using the dynamic linker provided in the Sourcery G++ sysroot, as described in Section 3.2.7.3, “Specifying the Sysroot Location at Runtime”. In other words, the command to invoke **gdbserver** in this case would be similar to:

```
> sysroot/lib/ld-linux.so.3 \
  --library-path sysroot/lib:sysroot/usr/lib \
  sysroot/usr/lib/bin/gdbserver :10000 program arg1 arg2 ...
```

3.2.8.2. Connecting to GDB Server from the Debugger

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

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

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

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

3.2.8.3. Setting the Sysroot in the Debugger

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

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

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

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

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

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

Refer to Section 3.2.7.1, “Installing the Sysroot” for more information about installing the sysroot on the target. Note that if you have installed a stripped copy of the provided libraries on the target, you should give GDB the location of an unstripped copy on the host.

3.3. Sourcery G++ Lite Release Notes

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

3.3.1. Changes in Sourcery G++ Lite 2009q1-203

Internal compiler error with Cortex-A9. A bug has been fixed that caused internal compiler errors when compiling with `-mcpu=cortex-A9`.

3.3.2. Changes in Sourcery G++ Lite 2009q1-200

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.

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.

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

Fix for `backtrace` function in dynamically-linked executables. The `backtrace` function now reports backtraces for dynamically-linked executables. Previously, this function worked only with statically-linked executables, i.e., executables linked with `-static`.

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

3.3.3. Changes in Sourcery G++ Lite 2009q1-176

Assembler fix for `-mauto-it`. A bug in the assembler that caused incorrect assembly of branches has been fixed. The bug only occurred when automatically generating IT instructions using `-mauto-it`.

Fix for `dlopen` in statically-linked executables. A bug has been fixed that made statically-linked programs crash when calling `dlopen` with `RTLD_GLOBAL` to request global availability of symbols from the shared module.

Overlay sections. **arm-none-linux-gnueabi-readelf** now correctly recognizes section headers for `ARM_DEBUGOVERLAY` and `ARM_OVERLAYSECTION` sections.

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

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.

Optimized `memcpy`. The implementation of `memcpy` has been optimized to increase performance on ARM targets that support prefetch instructions.

Optimized memory and string routines. The implementations of `memcpy`, `memset`, `strcmp`, `strcpy`, and `strlen` have been optimized to increase performance on ARM targets.

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.

Support for GLIBC `backtrace` function. Sourcery G++ Lite now supports the GLIBC `backtrace` function on ARM GNU/Linux targets. For more information, see Section 3.2.3, “GLIBC Backtrace Support”.

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.

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.

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.

arm-none-linux-gnueabi-objcopy bug fix. A bug has been fixed that caused **arm-none-linux-gnueabi-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.

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

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.

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

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.

ARM EABI attributes. An assembler bug that resulted in some object files generated from hand-written assembly being incorrectly tagged as using VFP instructions has been fixed. This tagging was harmless for objects linked with Sourcery G++ tools and libraries, but may have affected third-party tools and libraries.

3.3.4. Changes in Sourcery G++ Lite 2009q1-123

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.

ARMv4T library selection. Compiler options such as `-mcpu=arm740t`, specifying an ARMv4T CPU, now cause the compiler to link with the ARMv4T libraries without requiring `-march=armv4t` to be specified as well.

Core files in GDB. A bug in GDB has been fixed that caused incorrect values for CPSR (the status register) to be displayed when debugging core files. In some cases, this bug could also cause the **gcore** command to crash GDB.

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.

Support for VFP and Advanced SIMD (NEON) register display. The Sourcery G++ debugger can now display VFP and Advanced SIMD registers when debugging Linux applications with **gdb-server**. This may require a kernel patch; see Section 3.2.1, “Target Kernel Requirements” for details.

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

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.

VFP ABI support. Sourcery G++ now supports the VFP variant of the Procedure Call Standard for the ARM® Architecture (AAPCS) in addition to the default soft-float ABI. The VFP ABI uses VFP registers to pass function arguments and return values, resulting in faster floating-point code. Code compiled with the VFP ABI is not compatible with the soft-float ABI libraries provided with Sourcery G++ Lite; however, VFP ABI libraries for little-endian ARM v7-A processors are now available as add-ons for Sourcery G++ Professional Edition. For further information about floating-point compiler, ABI and library support in Sourcery G++, refer to Section 3.2.4.1, “Enabling Hardware Floating Point”.

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.

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.

3.3.5. Changes in Sourcery G++ Lite 2009q1-117

Thread cancellation in `aio_suspend`. A bug in the Thumb-2 version of `aio_suspend` has been fixed that caused programs to crash if the calling thread was canceled.

Linking big-endian programs for ARMv7-A. When linking for ARMv7-A targets with `-mbig-endian`, Sourcery G++ now implicitly assumes BE8 mode, rather than BE32.

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

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.

ARM atomic memory operations. Support has been added for atomic memory operations in ARM Linux applications via built-in functions (for example, `__sync_fetch_and_add`). Please refer to the GCC manual (Atomic Builtins) for further information.

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 GNU/Linux targets. This includes processors for which Sourcery G++ Lite does not include appropriate run-time libraries. In addition, CS3 support files for boards and processors that do not have an MMU to support running the Linux kernel have been removed. 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.

`timer_delete` bug. A bug has been fixed that caused some programs to crash or hang after calling `timer_delete`.

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. This affected thread-related code in the Thumb-2 versions of the standard C++ library provided with Sourcery G++, causing some multithreaded C++ programs to crash.

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. This affected code in the libraries provided with Sourcery G++ that handles `cleanup` attributes and POSIX thread cancellation.

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.

`sched_setaffinity` bug fix. A bug has been fixed that caused the Thumb-2 version of `sched_setaffinity` to fail inappropriately with `EFAULT`.

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.

3.3.6. Changes in Sourcery G++ Lite 2008q3-72

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.

Bug fix for `libSegFault.so`. Binaries linked with `libSegFault.so` now produce a correct backtrace when a segmentation fault occurs. Formerly, the location of the fault was shown incorrectly.

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.

3.3.7. Changes in Sourcery G++ Lite 2008q3-41

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.

Thump-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.

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.

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 3.2.4.3, "Half-Precision Floating Point".

3.3.8. Changes in Sourcery G++ Lite 2008q3-13

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.

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`. In addition, on GNU/Linux systems, the ARM C++ ABI utility functions are now exported from the `libstdc++.so` shared library. 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 GNU/Linux is now based on GCC version 4.3.2. For more information about changes from GCC version 4.2 that was included in previous releases, see `http://gcc.gnu.org/gcc-4.3/changes.html`.

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.

3.3.9. Changes in Sourcery G++ Lite 2008q1-126

Disassembler bug fix. A bug in the disassembler has been fixed that formerly caused **objdump** to crash when processing raw binary files, or other executables with an empty symbol table.

NEON assembler symbols. An assembler bug that caused spurious undefined symbols to be generated has been fixed. The `mov d0, d1` instruction would incorrectly cause an undefined symbol `d1` to be created.

Profiling bug fix. A bug that caused incorrect behavior and crashes when running code compiled for **gprof** profiling (`-pg`) has been fixed.

GDB info registers crash fix. Executing **info registers** after executing **flushregs** no longer crashes GDB.

3.3.10. Changes in Sourcery G++ Lite 2008q1-102

GDB and Ctrl+C on Windows . GDB no longer crashes when you press **Ctrl**+**C** twice during remote debugging to give up waiting for the target.

ARM Cortex-A9 processor support. The compiler can now generate code optimized for the ARM Cortex-A9 processor. This is enabled by the the `-mcpu=cortex-a9` command-line option.

MOVW and MOVT relocations. A linker error that resulted in incorrect offsets when processing relocations on `MOVW` and `MOVT` instructions referencing mergeable string sections has been fixed.

Improved argument-passing code. The compiler can now generate more efficient code for certain functions whose arguments must be sign-extended to conform with language or ABI conventions. The required conversion was formerly being performed both in the called function and at all call sites; now the redundant conversion has been eliminated for functions that can only be called within the compilation unit where they are defined.

Multi-process mode for gdbserver. The **gdbserver** utility has a new command-line option, `--multi`, that allows you to use it to debug multiple program instances. Refer to the Debugger manual for more information.

GDB `qOffsets` crash fix. GDB no longer crashes when a remote stub provides load offsets for an unlinked object file.

Linker error allocating ELF segments. A bug where the linker produces an incorrect error message with segments at the top of the address space has been fixed.

GCC stack size limit increased. On Windows hosts, the maximum stack size for the GCC executable has been increased. This means that more complex programs can be compiled.

Stack overflows in `printf`. Some stack overflows in `printf`-family functions have been fixed. These overflows occurred with format strings with very large precision values such as `%1.300000000s`, and with some invalid format strings.

Invalid object file after strip. A bug in the assembler has been fixed that formerly caused `.set` *`symbol expression`* constructs to emit *`symbol`* in the wrong section. This in turn caused inconsistent behavior after stripping the symbol table.

GCC update. The GCC package has been updated to version 4.2.3. This version includes numerous bug fixes since GCC 4.2.

License checking on Linux. Sourcery G++'s license-checking logic now includes a workaround for a kernel bug present in some versions of Linux. This bug formerly caused failures with an error message from the `cs-license` component.

Cortex-R4F and VFPv3-D16. Sourcery G++ now supports the ARM Cortex-R4F CPU and the VFPv3-D16 floating-point coprocessor. These can be selected with `-mcpu=cortex-r4f` and `-mfpu=vfpv3-d16`, respectively.

Overlapping operands for long multiply instructions. An incorrect assembler warning has been removed in the case of overlapping source and destination operands for `UMULL`, `SMULL`, `UMLAL` and `SMLAL` instructions on ARMv6 processors.

Size optimization bug. A code generation bug that caused corruption of function arguments when compiling with `-Os` has been fixed. The corruption occurred as part of the sibling call optimization.

C++ library ABI fix. GCC 4.2.1's `std::type_info` was not fully compatible with earlier versions. The ordering of four virtual functions has been fixed in this update.

Improved gdbserver thread support. The GNU/Linux remote debug agent, **gdbserver**, no longer fails when two threads call `pthread_create` at the same time. It also supports limited thread debugging when symbols for `libpthread.so` are not available.

GDB support for user-defined prefixed commands. The GDB **define** and **document** commands, which allow you to add new commands to the GDB command-line interface, now support creating commands within an existing prefix such as **target**. Hooks for prefixed commands are also supported. Refer to the Debugger manual for more information.

GDB update. The included version of GDB has been updated to 6.7.20080107. This update includes numerous bug fixes.

UNC pathname bug fix. A bug has been fixed that caused linker errors on Windows hosts when running a Sourcery G++ toolchain installed in a UNC path (`\\`*`host`*`\`*`directory`*).

Linker crash on invalid input files. Some older versions of GCC generated object files with invalid mergeable string sections when compiling with `-fmerge-all-constants`. This bug was fixed in Sourcery G++ as of version 4.1-43. However, since system libraries included with some GNU/Linux distributions were affected by this bug, the linker has now been changed to accept object files with such invalid sections, rather than crash or produce an error message.

GDB search path bug fix. A bug in GDB has been fixed that formerly resulted in an internal error when setting `solib-search-path` or `solib-absolute-prefix` after establishing a connection to a remote target.

Binutils update. The binutils package has been updated to version 2.18.50.20080215 from the FSF trunk. This update includes numerous bug fixes.

Race fixes in `setuid`. Several bugs in multi-threaded `setuid` have been fixed. The bugs led to threads with incorrect privileges and hangs at thread exit. The `setgid`, `seteuid`, `setegid`, `setreuid`, `setregid`, `setresuid`, and `setresgid` functions were also affected.

gdbserver support for execution wrappers. **gdbserver** has a new command-line option, `--wrapper`, which specifies a wrapper for any programs run by **gdbserver**. The specified wrapper can prepare the system and environment for the new program.

Read-only variables. The C++ compiler now places variables whose types are instantiations of template classes in a read-only data section if they are declared `const` and initialized with a constant value. This changes reduces the RAM usage of affected applications.

CodeSourcery Common Startup Code Sequence. Support for CS3, a unified startup scheme is included.

3.3.11. Changes in Sourcery G++ Lite 2007q3-51

Volatile postincrement and postdecrement bug fix. A code generation bug that caused postincrement or postdecrement of a volatile object to reread the modified value from that object in some contexts has been fixed. The bug affected code performing a comparison of the postincrement or postdecrement expression with a constant, or that was optimized to comparison with a constant.

Support for debugging with FlashPro3. Support has been added for debugging with the Actel FlashPro3 JTAG device on Windows hosts. This works only with Actel Cortex-M1 FPGAs.

C++ class debug information. The flag `-femit-class-debug-always` is now disabled by default. The flag produces duplicate C++ class debug information as a work-around for older debuggers.

Improved breakpoints in constructors and template functions. GDB now supports breakpoints on source code locations that have several code addresses associated with them. Setting a breakpoint on a constructor automatically associates the breakpoint with all constructor bodies generated by GCC. If you set a breakpoint on a line of a templated function, GDB breaks at the indicated line in all instantiations of the templated function.

GDB printf %p. GDB's **printf** command now supports the `"%p"` format specifier.

Widening multiply instructions for ARMv6 and later. GCC now makes use of the 32-to-64-bit widening multiply instructions (`umull`, `smull`, `umlal`, and `smlal`) when generating code for ARMv6 and later. A bug had caused these instructions to be used for ARMv3 to ARMv5 only.

GDB update. The included version of GDB has been updated to 6.6.20070821. This update includes numerous bug fixes.

Assembler code file name suffixes. GCC now recognizes `.sx` as well as `.S` as a file name suffix indicating assembler code which must be preprocessed. The alternate suffix may be useful in conjunction with other program development tools on Windows that do not distinguish case on filenames and treat `.S` the same as `.s`, which GCC uses to indicate assembler code without preprocessing.

3.3.12. Changes in Sourcery G++ Lite 2007q3-33

Preprocessing assembly code. The compiler driver passes `-I` options to the assembler, so that `#include` directives (processed by the preprocessor) and `.include` directives (processed by the assembler) use the same search path.

Dynamically-initialized `const` variables. Dynamically-initialized namespace-scope C++ variables are no longer placed in read-only data sections, even when marked `const`. These variables must be modified at startup, so they cannot be placed in ROM, even though their values cannot change once initialized.

Register allocation bug fix. A register allocation bug has been fixed. Under rare circumstances, the bug caused incorrect code generation.

iWMMXt bug fix. A GCC bug affecting code generation for iWMMXt processors has been fixed. The bug caused internal compiler errors when compiling some functions with large stack frames.

NEON coprocessor system registers. The assembler now accepts the `MVFR0` and `MVFR1` coprocessor registers in `fmrx` and `fmxr` instructions.

Disabling diagnostics for use of system header and library directories. The warnings for use of options such as `-I/usr/include` when cross compiling can be disabled with a new option `-Wno-poison-system-directories`. This option is intended for use in chroot environments when such directories contain the correct headers and libraries for the target system rather than the host.

Default linker script. GCC no longer uses the simulator linker script by default. To avoid a link failure, you must specify a linker script explicitly with the `-T` command-line option, or via the `Properties` item on the `Project` menu in the Sourcery G++ IDE.

Debugging of statically-linked threaded programs. GDB and EGLIBC now support thread debugging when using GCC's `-static` option. Existing statically-linked programs must be relinked after upgrading EGLIBC for this fix.

Thumb-2 doubleword writeback addressing modes. An assembler bug that caused writeback addressing modes for `ldrd` and `strd` to be incorrectly encoded has been fixed.

Stricter check for anonymous unions. G++ now issues an error about invalid code that uses the same name for a member of an anonymous union and an entity in the surrounding namespace. For example, you will now get an error about code like:

```
int i;
static union { int i; };
```

because both the global variable and the anonymous union member are named `i`. To make this code valid you must change one of the declarations to use a different name.

GCC update. The GCC package has been updated to version 4.2.1. This version includes numerous bug fixes since GCC 4.2.

Smaller code for C++ destructors. G++ now generates more compact code to handle the destruction of C++ objects declared at namespace scope or declared within a function scope using the `static` keyword.

Robustness on Microsoft Windows. Defects that sometimes caused GDB to become non-responsive on Microsoft Windows have been eliminated.

Debug information for thread-local variables. GCC now generates accurate debug information for thread-local variables.

Binutils update. The binutils package has been updated to the 2007-08-19 version of the pre-2.18 FSF trunk. This contains many new improvements and bug fixes. For more information, refer to the manuals for the individual utilities, and to the binutils web site at `http://www.gnu.org/software/binutils/`.

Debugging information fix. GCC no longer generates invalid debugging information for sections with no contents. The invalid debugging information caused the GNU/Linux prelinker to crash.

Calls to undefined weak symbols. The linker now implements semantics that comply to the ARM EABI for `R_ARM_CALL` and `T_ARM_THM_CALL` relocations against undefined weak symbols. These now result in a jump to the next instruction.

Thread synchronization no longer uses `swp`. All of the thread synchronization functions (such as `pthread_mutex_lock`) have been changed to avoid using the `swp` instruction. This instruction is deprecated in ARM V6 and executes very slowly on recent CPUs. Instead, the synchronization functions now use kernel-provided instruction sequences that are optimal for the processor in use.

Thumb-2 shift instruction aliases. The assembler now accepts `mov` with shifted operands as an alias for Thumb-2 shift instructions. For example `mov r0, r1, lsl r2` is encoded as `lsl r0, r1, r2`.

Thumb-2 branches to shared libraries. The linker can now generate PLT stubs for `R_ARM_THM_JUMP24` and `R_ARM_THM_JUMP19` relocations. This occurs when Thumb-2 branch instructions refer to symbols imported from shared libraries.

Inlined function debugging fix. GDB now backtraces correctly when stopped at the first instruction of an inlined function. Earlier versions would sometimes encounter internal errors in this situation.

Assembler skipping \ characters. A bug is fixed where the assembler would skip \ characters when they appeared at certain positions in the input file. This bug primarily affected assembler macros.

Improved diagnostics for region overflow. The linker will now give more helpful diagnostics when the object files being linked are too big for one of the memory regions defined in the linker script.

EABI object attribute merging. The linker now properly merges EABI object attributes into its output file.

Thumb-2 exception return instructions. An assembler bug that caused `subs pc, lr, #const` and `movs pc, lr` to be incorrectly encoded has been fixed.

Tag_ABI_PCS_wchar_t object attributes. Objects generated with `-fshort-wchar` are now given the correct `Tag_ABI_PCS_wchar_t` EABI object attribute annotations.

Spurious compiler warnings eliminated. GCC no longer emits warnings when linker-specific command-line options are provided in combination with modes that do not perform linking, such as with the `-c` flag.

Debugging of inlined functions. GDB now supports inlined functions. GDB can include inlined functions in the stack trace; display inlined functions' arguments and local variables; and step into, over, and out of inlined functions.

Uppercase special register names. The assembler now accepts both uppercase and lowercase special register names when assembling `msr` and `mrs` instructions for the Microcontroller profile of the ARM Architecture.

Debugger access to out-of-bounds memory. GDB turns on `inaccessible-by-default` by default, disallowing access to memory outside the regions specified in a board configuration.

Call shortening bug fix. GCC no longer overrides `__attribute__((long_call))` on calls to locally-defined functions when the function is weak, or when it is in a different section from the caller.

Binutils update. The binutils package has been updated from version 2.17 to the pre-2.18 FSF trunk. This is a significant update with many improvements and bug fixes.

Changes to the assembler (**as**) include:

- On MIPS targets, support for additional processors and the SmartMIPS and DSP Release 2 extensions has been added.

New linker (**ld**) features include:

- A new command-line option `--default-script` has been added to give more precise control over linker script processing.
- There are new command-line options `-Bsymbolic-functions`, `--dynamic-list`, `--dynamic-list-cpp-new`, and `--dynamic-list-data` to control symbols that should be dynamically linked.
- The new `--print-gc-sections` option lists sections removed by garbage collection.

Other changes include:

- The **objcopy** utility has a new `--extract-symbol` option to extract only symbol table information from the input file.
- The **gprof** utility now allows input files to have histogram records for several memory ranges, provided those ranges are disjoint.

For more information, refer to the manuals for the individual utilities, and the binutils web site at `http://www.gnu.org/software/binutils/`.

GDB update. The included version of GDB has been updated to 6.6.50.20070620. This update includes numerous bug fixes.

Thread cancellation bug fix. A bug in EGLIBC causing thread cancellation to fail when running under Linux kernels 2.6.18 and later has been fixed.

Forced alignment of array variables. A new option `-falign-arrays` has been added to the compiler. Specifying this option sets the minimum alignment for array variables to be the largest power of two less than or equal to their total storage size, or the biggest alignment used on the machine, whichever is smaller. This option may be helpful when compiling legacy code that uses type punning on arrays that does not strictly conform to the C standard.

ARM EABI compliance. Objects produced by Sourcery G++ are now marked as ARM ELF version 5 rather than ARM ELF version 4. This reflects compliance with recent revisions of the ARM EABI. Sourcery G++ still accepts objects marked with version 4.

Smaller C++ applications. The C++ runtime library has been modified so that using namespace-scope objects with destructors does not pull in unnecessary support functions. Therefore, statically linked C++ applications compiled with `-fno-exceptions` are substantially smaller.

ARMv6-M floating-point bug fix. A bug affecting conversion of wider floating-point types to subnormal `float` values on ARMv6-M processors has been fixed.

3.3.13. Changes in Sourcery G++ Lite 2007q1-21

NEON coprocessor system registers. The assembler now accepts the `MVFR0` and `MVFR1` coprocessor registers in `fmrx` and `fmxr` instructions.

Disabling diagnostics for use of system header and library directories. The warnings for use of options such as `-I/usr/include` when cross compiling can be disabled with a new option `-Wno-poison-system-directories`. This option is intended for use in chroot environments when such directories contain the correct headers and libraries for the target system rather than the host.

Thumb-2 doubleword writeback addressing modes. An assembler bug that caused writeback addressing modes for `ldrd` and `strd` to be incorrectly encoded has been fixed.

Thumb-2 shift instruction aliases. The assembler now accepts `mov` with shifted operands as an alias for Thumb-2 shift instructions. For example `mov r0, r1, lsl r2` is encoded as `lsl r0, r1, r2`.

Thumb-2 branches to shared libraries. The linker can now generate PLT stubs for `R_ARM_THM_JUMP24` and `R_ARM_THM_JUMP19` relocations. This occurs when Thumb-2 branch instructions refer to symbols imported from shared libraries.

EABI object attribute merging. The linker now properly merges EABI object attributes into its output file.

Thumb-2 exception return instructions. An assembler bug that caused `subs pc, lr, #const` and `movs pc, lr` to be incorrectly encoded has been fixed.

Tag_ABI_PCS_wchar_t object attributes. Objects generated with `-fshort-wchar` are now given the correct `Tag_ABI_PCS_wchar_t` EABI object attribute annotations.

Uppercase special register names. The assembler now accepts both uppercase and lowercase special register names when assembling `msr` and `mrs` instructions for the Microcontroller profile of the ARM Architecture.

3.3.14. Changes in Sourcery G++ Lite 2007q1-10

Disassembly of overlapping sections. A bug in the disassembler that caused code to be displayed as data (and vice-versa) in files with overlapping sections has been fixed. This mainly affects the **objdump** utility.

Installer hangs while refreshing environment. The Sourcery G++ installer for Microsoft Windows now updates the `PATH` environment variable without waiting for open applications to acknowledge the update. This change prevents open applications from blocking the installer's progress.

Improved assembler diagnostics for 8-bit offsets. The assembler now correctly diagnoses out-of-range offsets to instructions such as `LDRD` as 8-bit rather than half-word offsets.

Less disk space required for installation. Sourcery G++ Lite packages are smaller because multiple copies of files have been replaced with hard and/or symbolic links when possible. Both the size of the installer images and the amount of disk space required for an installed package have been reduced.

Thumb register corruption fix. A bug in the compiler that could cause register corruption in Thumb mode has been fixed. The compiler was formerly emitting code to restore registers on function return that was not interrupt safe.

__aeabi_lcmp. An error in the libgcc implementation of `__aeabi_lcmp` that caused incorrect results to be returned has been fixed. This is a support routine defined by the ARM EABI. GCC does not normally use this routine directly, however it may be used by third-party code.

The `\@` assembler pseudo-variable. A bug in the assembler that caused uses of the `\@` pseudo-variable to be mis-parsed as comments has been fixed.

Crash when generating vector code. A bug that sometimes caused the compiler to crash when invoked with the `-ftree-vectorize` option has been fixed.

Propagation of Thumb symbol attributes. Symbols referring to Thumb functions on ARM targets now have their Thumb attribute correctly propagated to any aliases defined with `.set` or `.symver`.

Linking of non-ELF images. A linker bug that could cause a crash when linking non-ELF objects for ARM targets has been fixed.

Invalid load instructions. A bug in the compiler which caused it to generate invalid assembly (e.g. `ldrd r0, [#0, r2]`) has been fixed.

VFPv3/NEON debug information. A bug in the compiler which caused it to generate incorrect debug information for code using VFPv3/NEON registers has been fixed. The debugger is now able unable to locate and display values held in these registers.

ARMv6-M system instructions. An assembler bug that caused some ARMv6-M system instructions to be incorrectly rejected has been fixed. The affected instructions are `msr`, `mrs`, `yield`, `wfi`, `wfe` and `sev`.

Assembly of Thumb-2 load/store multiple instructions. The Thumb-2 `ldm` and `stm` assembly mnemonics are now assembled to `ldr` and `str` instructions when a single register is transferred, as specified in the Thumb-2 Architecture Supplement.

Conditional Thumb-2 branch instructions. A linker bug that could cause objects involving conditional Thumb-2 branch instructions to be incorrectly rejected has been fixed.

Alignment bug fix. A bug has been fixed that formerly caused incorrect code to be generated in some situations for copying structure arguments being passed by value. The incorrect code caused alignment errors on stack accesses on some targets.

3.3.15. Changes in Sourcery G++ Lite 2007q1-3

Thumb-2 runtime libraries. Sourcery G++ now includes runtime libraries built as Thumb-2 code for use on ARMv7 systems. These can be found in the `libc/thumb2/` directory.

Marvell Feroceon support. Sourcery G++ Lite now generates code optimized for Marvell Feroceon CPUs when the `mcpu=marvell-f` option is specified. This option also selects runtime libraries optimized for this processor.

Assembly of SRS instructions. An assembler bug that resulted in incorrect encoding of the Thumb-2 `SRS` instruction has been fixed. In addition the assembler supports explicit specification of the base register, as accepted by other ARM toolchains.

VFP disassembly crash. A bug that caused crashes when disassembling some forms of the VFP `fmrx` and `fmxr` instructions has been fixed.

Improved debugging for optimized code. GDB's ability to print and change variables' values in optimized code is improved. GDB now tracks variable scopes more accurately, making better use of the detailed debugging information produced by Sourcery G++ compilers.

Improved handling of Windows paths in GDB. GDB now properly recognizes the names of source files that were passed to the compiler using an absolute path on Windows. You may refer to the file either by its base name (without any leading directory components), by the exact path passed to the compiler, or by its absolute path.

ARM Cortex-R4 performance improvements. Sourcery G++ Lite now generates faster code when compiling for the ARM Cortex-R4 processor by scheduling instructions for the processor's pipelines. To generate code for this processor, use the `-mcpu=cortex-r4` command-line option.

GDB update. The included version of GDB has been updated to 6.6.50.20070228. This update includes numerous bug fixes and improved support for C++ pointers to members.

Assembling Thumb store-multiple instructions. The assembler now issues an error message instead of crashing on load/store multiple instructions that incorrectly use Thumb-2 addressing modes (e.g., `ldmdb`) in legacy Thumb syntax mode. If you want to use these address modes, you should use unified syntax mode instead.

ARM Cortex-A8 performance improvements. Sourcery G++ Lite now generates faster code when compiling for the ARM Cortex-A8 processor by scheduling instructions for the processor's dual-issue pipelines. To generate code for this processor, use the `-mcpu=cortex-a8` command-line option.

GCC version 4.2. Sourcery G++ Lite for ARM GNU/Linux is now based on GCC version 4.2. For more information about changes from GCC version 4.1 that was included in previous releases, see `http://gcc.gnu.org/gcc-4.2/changes.html`.

Fix --gc-sections and C++ exceptions. A bug in the `--gc-sections` linker option has been fixed. Previously this would incorrectly remove unwinding tables, breaking C++ applications that use exceptions.

Symbols defined in linker scripts. A bug is fixed that caused the linker to crash in some circumstances when a linker script defined a symbol in an output section. Typically usage is where the script contained a `__DATA_LOAD = LOADADDR(.data)` statement in the `.data` section.

ARM NEON store intrinsics bug fix. A compiler bug that incorrectly caused calls to ARM NEON store intrinsics (such as `vst1_u8`) to be optimized away has been fixed.

Improvements to ARM NEON support. The ARM NEON support in GCC has been enhanced to comply with new rules for containerized vector types specified in the ARM procedure call standard. Additionally, the compiler now rejects implicit conversions between NEON polynomial vector types and NEON integer vector types of the same layout.

Complex numbers bug fix. A bug that could lead to incorrect code generation for code using complex numbers has been fixed.

Use of system header and library directories diagnosed. The compiler and linker now diagnose the incorrect use of native system header and library directories for cross-compilation. This typically arises from options such as `-I/usr/X11R6/include` hard-coded in build scripts written without a view to cross-compilation.

Initialization priorities. The `constructor` and `destructor` function attributes now accept an optional priority argument. Constructors with small priorities are run before those with larger priorities; the opposite is true for destructors. For example:

```
void f __attribute__((constructor(500)));
void f() {
  /* Perform initialization.  */
}
```

defines a function `f` with priority 500. This function will be run before constructors with larger priorities. Constructor and destructors with no explicit priority argument have priority 65535, the maximum permitted value.

Thumb-2 IT block code generation error fixed. A bug in Thumb-2 code generation has been fixed. This bug would result in missing IT instructions, causing the assembler to reject the code.

iWMMXt compiler errors. A compiler bug that caused invalid assembly when generating iWMMXt code has been fixed.

Thumb-2 stack decrement misassembly. An assembler bug that resulted in incorrect encoding of the 32-bit Thumb-2 form of the `sub sp, sp, #const` instruction has been fixed. Previously this was misassembled as `subs`.

Naked functions. Functions marked with `__attribute__((naked))` no longer contain prologue and epilogue code. Please refer to the GCC manual for the proper use of this attribute.

Destructor execution order. Prioritized destructors (whether specified with the `destructor` function attribute in C or the `init_priority` variable attribute in C++) are now executed in the correct order. Previous releases ran the destructors in an indeterminate order.

Fix addr2line defect. The binary utility **addr2line** now operates correctly on 64-bit targets with DWARF2 debug information.

Thumb-2 assembler infinite loop. An assembler bug that would cause it to enter an infinite loop when processing some Thumb-2 assembly has been fixed.

Assembler warnings about overlapping multiplication operands. The assembler no longer warns about overlapping `Rd` and `Rm` operands when assembling `mul` and `mla` instructions for the ARM architecture version six or above.

Improve handling of corrupt debug information. The binary utility **readelf** now copes more gracefully with corrupted DWARF 2 information.

Smaller C++ programs. Rarely-used functions in the C++ runtime library have been isolated into separate object files so that they will not be included unless needed. As a result, most statically linked C++ programs are smaller.

3.3.16. Changes in Sourcery G++ Lite 4.1-37

Preserve volatile accesses. Reads from volatile memory are no longer incorrectly optimized away at higher optimization levels.

3.3.17. Changes in Sourcery G++ Lite 4.1-34

Implicit conversions between generic vector types. Implicit conversions between generic vector types are now only permitted when the two vectors in question have the same number of elements and compatible element types. (Note that the restriction involves *compatible* element types, not implicitly-convertible element types: thus, a vector type with element type `int` may not be implicitly converted to a vector type with element type `unsigned int`.) This restriction, which is in line with specifications for SIMD architectures such as AltiVec, may be relaxed using the flag `-flax-vector-conversions`. This flag is intended only as a compatibility measure and should not be used for new code.

`type_info` comparison fix. Comparison of `type_info` objects now uses pointer comparison where possible.

C++ forced unwinding fixes. Some bugs relating to forced unwinding through C++ code have been fixed.

Support for additional Stellaris boards. Linker scripts are provided for the 6xx and 8xx series Stellaris boards.

Linux support for USB Debug Sprite. A new driver is included to allow the Sourcery G++ Lite USB Debug Sprite to run on Linux hosts. See Chapter 3, *Sourcery G++ Lite for ARM GNU/Linux* for additional information.

3.3.18. Changes in Sourcery G++ Lite 4.1-33

Linker scripts. A bug is fixed where an erroneous linker script would cause a linker crash. An error message is now produced.

Newlib memory use improvements. The memory overhead of linking with newlib is reduced. Applications that use only a minimal set of library features may now require significantly less memory.

3.3.19. Changes in Sourcery G++ Lite 4.1-31

Compiler alias analysis. The type-based alias analysis performed by the compiler when compiling with `-O2` or with `-fstrict-aliasing` is now more conservative. The more aggressive analysis used in previous versions sometimes resulted in incorrect code generation.

Fully relocatable preprocessor. When cross-compiling, the default preprocessor search path includes only the directories present in the installed toolchain. This speeds up the preprocessor and prevents the unintentional use of unrelated files and directories on the machine where it is installed.

3.3.20. Changes in Sourcery G++ Lite 4.1-29

Support for new-style symbol hashing. Support has been added in binutils and the prelinker for new-style (also known as `DT_GNU_HASH`) symbol hashing. This can dramatically speed up symbol resolution time and is particularly applicable in environments where full prelinking is not possible (for example where shared libraries are dynamically opened at runtime). The new-style hashing may be enabled by passing `--hash-style=gnu` to the linker.

Prelinker update. The prelinker has been updated to the current upstream sources and some bugs affecting operation have been fixed.

3.3.21. Changes in Sourcery G++ Lite 4.1-28

Improved support for ROM debugging. GDB now determines ROM regions automatically from the memory map included in target configuration files. This information is used to determine when hardware breakpoints should automatically be used (for instance the **step**, **next** and **finish** commands). Separate ROM configurations have been removed from the Eclipse debugger menu. The Eclipse GUI has been extended to provide improved support for debugging programs in ROM, when a memory map is not automatically available.

3.3.22. Changes in Sourcery G++ Lite 4.1-27

Rename Windows executables. The Windows host tools **make.exe** and **rm.exe** are now named **cs-make.exe** and **cs-rm.exe**. This change avoids conflicts with tools provided by other distributors.

iWMMXt bug fixes. Some bugs involving incorrect code generation and internal compiler errors when generating iWMMXt code have been fixed.

Cortex-M3 startup code. The ARMv7-M startup code (`armv7m-crt0.o`) incorrectly contained ARM code. This has been replaced with Thumb-2 code.

ARM EABI coverage testing support. Coverage testing using GCOV is now supported for the ARM EABI target. Please refer to the *GNU C Compiler Manual (HTML)* for more information on coverage testing.

3.3.23. Changes in Sourcery G++ Lite 4.1-23

Windows debugging fix. In recent releases of Sourcery G++ Lite, the GDB **target remote |** command would hang on Windows. This affected both command line and Eclipse debugging when using the Sourcery G++ Lite Debug Sprite.

Stellaris USB Debug Sprite improvements. The former USB Debug Stub, **armswd**, is now known as the USB Debug Sprite, and has been renamed to **arm-stellaris-eabi-sprite**. In addition, its initialization sequence has been updated to recognize the r1p1 release of the Cortex-M3 processor.

Incompatible changes to Stellaris linker scripts. Sourcery G++ Lite now supports linking executables to run from RAM as well as ROM. As part of this change, there are now separate RAM and ROM versions of the linker scripts for each supported board, and the former ROM-based versions have been renamed. For example, if you were formerly linking with `-T lm3s10x.ld`, you should now use `-T lm3s10x-rom.ld` to get the same behavior.

3.3.24. Changes in Sourcery G++ Lite 4.1-21

Eclipse debuggers. Eclipse configurations for debugging arm-none-eabi applications using the GDB simulator and remote debug stubs have been added.

iWMMXt2 support. The assembler and disassembler now support iWMMXt2 instructions.

NEON intrinsics support. GCC now supports NEON intrinsics defined in the `arm_neon.h` header. These are the same intrinsics supported by the ARM RealView® compiler and are documented in the 'ARM NEON Intrinsics' section of the GCC manual.

3.3.25. Changes in Sourcery G++ Lite 4.1-19

ARMv4t linux multilib. Linux configurations now support ARMv4t CPUs.

Linker scripts. Several problems with the linker scripts for bare-metal targets have been fixed.

3.3.26. Changes in Sourcery G++ Lite 4.1-18

Binutils update. The binutils in this release is based on the final binutils 2.17 release.

GDB update. The included version of GDB has been upgraded to 6.5.50.20060822. This includes numerous bug fixes from the previous version.

GDB support for flash memory. The GDB **load** command can now write to flash memory, if the remote debugging stub contains appropriate support.

Compiler support for NEON. Initial GCC support for autovectorization and generation of NEON SIMD instructions has been added.

Bare metal Cortex-M3 configurations. Bare metal configurations now support generating images for use on ARMv7-M devices (eg. Cortex-M3).

iWMMXt support in GLIBC. GLIBC's `setjmp` and `longjmp` now support saving and restoring iWMMXt registers on hardware with those registers. This requires a kernel reporting `iwmmxt` in the `Features` entry in `/proc/cpuinfo`.

iWMMXt exception handling support. Exception handling now restores the values of iWMMXt registers correctly.

Corrected IPC functions. A bug in GLIBC's `msgctl`, `semctl`, and `shmctl` functions has been corrected.

3.3.27. Changes in Sourcery G++ Lite 4.1-16

GCC update. This release is based on GCC 4.1.1.

Fully relocatable compiler. The compiler now searches for its components only in the directory where it has been installed, and no longer also searches pathnames matching the directory where it was configured. This speeds up the compiler and prevents problems with unintentionally finding unrelated files or directories on the machine where it has been installed.

Stack permission marking for ARM GNU/Linux. Non-executable stacks can provide increased security against some forms of buffer overflow attacks. The tools involved must coordinate the annotation of required stack permissions, either executable, or non-executable. For ARM GNU/Linux targets the compiler now outputs annotations indicating the required stack permissions.

3.3.28. Changes in Sourcery G++ Lite 4.1-15

Stabs debugging information support. Using the Stabs debugging format (available with `-gstabs` or `-gstabs+`) now works in conjunction with `-mthumb`. CodeSourcery recommends the default DWARF debugging format (available with `-g`) as DWARF is a more comprehensive debugging format.

3.3.29. Changes in Sourcery G++ Lite 4.1-13

Stellaris linker scripts in IDE. Linker scripts may now be selected via a drop-down menu in Eclipse.

Stellaris linker scripts for 3xx series CPUs. The linker scripts for 3xx Series CPUs now place the ISR vector at address zero, as required by all Cortex-M3 cores.

Stellaris USB Debug Sprite improvements. Bug fixes and new features include:

- A bug that caused the stub not to correctly update the program counter and other register values was fixed. As a result of this fix, it is now possible to run programs residing in SRAM using the `continue` command from GDB.
- The stub no longer prints status messages via GDB console output when invoked with the `-q` command-line option.
- The stub's initialization sequence was updated to recognize revision C Cortex-M3 hardware.

3.3.30. Changes in Sourcery G++ Lite 4.1-9

Stellaris USB Debug Sprite improvements. Program images exceeding 4K can now be uploaded to flash memory.

Additional Stellaris boards supported. The Stellaris 301, 310, 315, and 316 CPUs are now supported. Linker scripts have been added for these boards.

3.3.31. Changes in Sourcery G++ Lite 4.1-8

Stellaris USB Debug Sprite improvements. Several bug fixes and enhancements were made to the USB Debug Stub. In particular:

- Bugs in the implementation of `open`, `read`, and `lseek` were fixed.
- Support was added for `isatty`, `rename`, `unlink`, and `system`.
- Memory reads that span 4K block boundaries now work correctly.

3.3.32. Changes in Sourcery G++ Lite 4.1-4

Runtime libraries. Support for ARMv7 including Cortex-M3 and pure Thumb-2.

Assembler. Support for NEON and VFPv3, including unified NEON/VFP syntax.

3.3.33. Changes in Sourcery G++ Lite 4.1-1

Initial release. This release is based on GCC 4.1.0.

Chapter 4
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.

4.1. 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-none-linux-gnueabi`.

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.

4.2. System Requirements

4.2.1. 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.

4.2.2. 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.

4.2.3. Target System Requirements

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

4.3. 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 4.4, “Installing Sourcery G++ Lite”.

If you have a Sourcery G++ subscription (or evaluation), then you can log into the Sourcery G++ Portal[1] to download your Sourcery G++ toolchain(s). CodeSourcery also makes some toolchains available to the general public from the Sourcery G++ web site[2]. These publicly available toolchains do not include all the functionality of CodeSourcery's product releases.

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.

4.4. 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.

4.4.1. 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.

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
```

4.4.2. 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
```

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

After the installer starts, follow the on-screen dialogs to install Sourcery G++ Lite.

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
```

4.4.3. 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++-2009q1`.

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
```

4.5. Installing Sourcery G++ Lite Updates

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

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 GNU/Linux target all begin with **arm-none-linux-gnueabi**. This means that you can install Sourcery G++ for multiple target systems in the same directory without conflicts.

4.6. 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.

4.6.1. 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 GNU/Linux`. Select `Change/Remove` and follow the on-screen dialogs to uninstall Sourcery G++ Lite.

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

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

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

4.6.2. 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-none-linux-gnueabi` 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.

4.6.3. 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.

4.7. Setting up the Environment

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

4.7.1. Setting up the Environment on Microsoft Windows Hosts

4.7.1.1. 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-none-linux-gnueabi-g++ -v
```

Verify that the last line of the output contains: `Sourcery G++ Lite 2009q1-203`.

4.7.1.2. 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.

4.7.2. 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-none-linux-gnueabi/man`.

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

```
> arm-none-linux-gnueabi-g++
```

and verifying that you receive the message:

```
arm-none-linux-gnueabi-g++: no input files
```

Chapter 5 Using Sourcery G++ from the Command Line

This chapter demonstrates the use of Sourcery G++ Lite from the command line. This chapter assumes you have installed Sourcery G++ Lite as described in Chapter 4, *Installation and Configuration*.

5.1. 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-none-linux-gnueabi, as indicated by the **arm-none-linux-gnueabi** command prefix.

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

```
#include <stdio.h>

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

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

Compile and link this program using the command:

```
> arm-none-linux-gnueabi-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-none-linux-gnueabi-gcc** with **arm-none-linux-gnueabi-g++**.)

5.2. Running Applications on the Target System

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

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

```
> factorial
```

You should see:

```
factorial(0) = 1
factorial(1) = 1
factorial(2) = 2
factorial(3) = 6
factorial(4) = 24
factorial(5) = 120
factorial(6) = 720
factorial(7) = 5040
factorial(8) = 40320
factorial(9) = 362880
```

5.3. 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.

5.3.1. 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 6, *Sourcery G++ Debug Sprite* for detailed information about the supported devices.

You can start the Sprite directly from within GDB:

```
(gdb) target remote | arm-none-linux-gnueabi-sprite arguments
```

Refer to Section 6.2, “Invoking Sourcery G++ Debug Sprite” for a full description of the Sprite arguments.

5.3.2. 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 GNU/Linux target. Follow the instructions in Chapter 3, *Sourcery G++ Lite for ARM GNU/Linux* to install and run **gdbserver** on your target system.

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

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

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

Chapter 6
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 GNU/Linux.

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

The Sprite acts as an interface between GDB and external debug devices and libraries. Refer to Section 6.2, "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 GNU/Linux* for details about how to install and run **gdbserver** on the target.

Important

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

6.1. 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-none-linux-gnueabi-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 2009q1-203)
armusb: [speed=<n:0-7>] ARMUSB device
  armusb:/// - ARMUSB Device
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 6.2, "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.

6.2. Invoking Sourcery G++ Debug Sprite

The Debug Sprite is invoked as follows:

```
> arm-none-linux-gnueabi-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 GNU/Linux:

`rdi`	Use an RDI debugging device. Refer to Section 6.4, “Remote Debug Interface Devices”.
`flashpro`	Use a FlashPro debugging device. Refer to Section 6.5, “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. 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 6.7, “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 6.8, “Board File Syntax” for more information.

6.3. 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-none-linux-gnueabi-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.

6.4. 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.

6.5. 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-none-linux-gnueabi-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.

6.5.1. 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-none-linux-gnueabi-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.

6.6. 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.

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-none-linux-gnueabi-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.

6.7. Supported Board Files

The Sourcery G++ Debug Sprite for ARM GNU/Linux includes support for the following target boards. Specify the appropriate *`board-file`* as an argument when invoking the sprite from the command line.

Board	**Config**
ARMulator (RDI)	`armulator`

6.8. 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-none-linux-gnueabi/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 7
Next Steps with Sourcery G++

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

7.1. Sourcery G++ Knowledge Base

The Sourcery G++ Knowledge Base is available to registered users at the Sourcery G++ Portal[1]. 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.

For more information on CodeSourcery support, see Chapter 2, *Sourcery G++ Subscriptions*.

7.2. 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-none-linux-gnueabi/` 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-none-linux-gnueabi/man/man1
```

Then you can invoke **man** as:

```
> man ./arm-none-linux-gnueabi-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 4.7, “Setting up the Environment” for instructions. Then you can invoke **man** with just the command name rather than a pathname.

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

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