Sourcery G++ Lite

ColdFire ELF

Sourcery G++ Lite 4.3-54

Getting Started

Sourcery G++ Lite: ColdFire ELF: Sourcery G++ Lite 4.3-54: Getting Started

CodeSourcery, Inc.

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.

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. If you are an administrator installing Sourcery G++ Lite on a UNIX-like system for all of your users to use, you should also be familiar with the package-management software (such as the Red Hat Package Manager) for your system.

2. Organization

This document is organized into the following chapters and appendices:

Chapter 1, *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 ColdFire ELF*	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, *CS3™: The CodeSourcery Common Startup Code Sequence*	CS3 is CodeSourcery's low-level board support library. This chapter describes the organization of the system startup code and tells you how you can customize it, such as by defining your own interrupt handlers. This chapter also documents the boards supported by Sourcery G++ Lite and the compiler and linker options you need to use with them.
Chapter 7, *Sourcery G++ Debug Sprite*	This chapter describes the use of the Sourcery G++ Debug Sprite for remote debugging. The Sprite allows you to debug programs running on a bare board without an operating system. This chapter includes information about the debugging devices and boards supported by the Sprite for ColdFire ELF.

Chapter 8, *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 ColdFire ELF 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]
Sourcery G++ Debug Sprite for ColdFire	CodeSourcery License
CCS Server	CCS Server License
Newlib C Library	Newlib License [4]
GNU Make	GNU General Public License 2.0 [5]
GNU Core Utilities	GNU General Public License 2.0 [6]

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://sources.redhat.com/newlib/COPYING.NEWLIB
[5] http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
[6] 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.

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

5.2. **Sourcery G++ Debug Sprite for CCS Debugging Devices.** The Sourcery G++ Debug Sprite for CCS includes the CodeWarrior Connection Server Dynamic Linked Library (“CCS DLL”) from Freescale Semiconductor, Inc. You may use the CCS DLL only in conjunction with Sourcery G++ on a Windows or Linux-hosted platform. You may not translate, reverse engineer, decompile, or disassemble the CCS DLL, except to the extent applicable law specifically prohibits such restriction. If You are a U.S. Government end user, the CCS DLL is “restricted computer software” and is subject to FAR 52.227-19(c)(1) and (c)(2).

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

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

`--with-bugurl` configuration options that embed CodeSourcery trademarks in the resulting binary.

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

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

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

11. **Warranty Disclaimer; Limitation of Liability.** CODESOURCERY AND ITS LICENSORS PROVIDE THE SOFTWARE "AS-IS" AND PROVIDED WITH ALL FAULTS. CODESOURCERY DOES NOT MAKE ANY WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. CODESOURCERY SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SYSTEM INTEGRATION, AND DATA ACCURACY. THERE IS NO WARRANTY OR GUARANTEE THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED, ERROR-FREE, OR VIRUS-FREE, OR THAT THE SOFTWARE WILL MEET ANY PARTICULAR CRITERIA OF PERFORMANCE, QUALITY, ACCURACY, PURPOSE, OR NEED. YOU ASSUME THE ENTIRE RISK OF SELECTION, INSTALLATION, AND USE OF THE SOFTWARE. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS AGREEMENT. NO USE OF THE SOFTWARE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

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

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

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

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

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

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

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

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

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

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

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

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 ColdFire ELF

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 build a target application, Sourcery G++ automatically selects the multilib matching the build options you have selected.

Note that a given multilib may be compatible with additional processors and configurations beyond those explicitly named here.

The following library configurations are available in Sourcery G++ Lite for ColdFire ELF.

MCF5206	
Command-line option(s):	default

MCF51QE	
Command-line option(s):	`-mcpu=51qe`

MCF5206E	
Command-line option(s):	`-mcpu=5206e`

MCF5208	
Command-line option(s):	`-mcpu=5208`

MCF5307	
Command-line option(s):	`-mcpu=5307`

MCF532X/7X	
Command-line option(s):	`-mcpu=5329`

MCF5407	
Command-line option(s):	`-mcpu=5407`

MCF54455	
Command-line option(s):	`-mcpu=54455`

MCF547X/8X	
Command-line option(s):	`-mcpu=5475`

MCF547X/8X - Soft-Float	
Command-line option(s):	`-mcpu=5475 -msoft-float`

3.2. Using Flash Memory

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

1. You must use an appropriate linker script that identifies the ROM memory region on your target board, and locates the program text within that region. Refer to Chapter 6, *CS3™: The CodeSourcery Common Startup Code Sequence* for information about the boards supported by Sourcery G++.

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

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

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

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

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

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 4.3-54

ColdFire 51QE support. CS3 now includes support for MCF51QE128, MCF51QE64 and MCF51QE32 boards.

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.

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

Sprite 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.

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

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

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

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

Support for MCF5301x processors. Support has been added for the ColdFire MCF5301x (Longjin) family of microprocessors. To compile for the MCF53011, MCF53012, MCF53013, MCF53014, MCF53015, MCF53016 or MCF53017 processor, use `-mcpu=53011`, `-mcpu=53012`, `-mcpu=53013`, `-mcpu=53014`, `-mcpu=53015`, `-mcpu=53016` or `-mcpu=53017`, respectively.

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

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

Support for MCF5225x processors. Support has been added for the ColdFire MCF5225x (Kirin3) family of microprocessors. To compile for the MCF52252, MCF52254, MCF52255, MCF52256, MCF52258 or MCF52259 processor, use `-mcpu=52252`, `-mcpu=52254`, `-mcpu=52255`, `-mcpu=52256`, `-mcpu=52258` or `-mcpu=52259`, respectively.

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

Support for MCF51AC, MCF51CN and MCF51EM processors. Support has been added for the ColdFire MCF51ACnn (Celis), MCF51CNnn (Lasko) and MCF51EM (Nucleus) families of microprocessors. To compile for these processors use `-mcpu=51ac`, `-mcpu=51cn` or `-mcpu=51em` options respectively.

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

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

3.3.2. Changes in Sourcery G++ Lite 4.3-11

Newlib manuals. The documentation packaged with Sourcery G++ Lite now includes the Newlib C Library and Math Library manuals.

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

Coldfire cache initialization. The CS3 startup code now correctly initializes the `CACR` register to enable caching for V3, V4 and V4e cores. V2 cores are already initialized correctly. Only write through data caching is enabled on V4 cores, while both data and instruction caches are enabled on the other cores.

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

Placing bss-like regions in load regions. The linker no longer issues an incorrect error message when a bss-like section is placed at specific load region. The linker formerly incorrectly considered the section as taking up space in the load region.

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

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

GCC version 4.3.2. Sourcery G++ Lite for ColdFire ELF 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`.

ColdFire flash module. A bug is fixed in linker scripts for ColdFire flash modules. The flash module configuration block was previously omitted, leading to access errors in some regions of the flash memory.

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.

`SIZE_MIN` and `SIZE_MAX`. `SIZE_MAX` is now correctly defined in `stdint.h` to be an unsigned quantity equal to the maximum possible value of a `size_t`, as required by the C standard. In addition, the definition of `SIZE_MIN` has been removed, as this constant is not prescribed by the C standard.

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.

Support for MCF5227x processors. Support has been added for the ColdFire MCF5227x (DragonFire0) family of microprocessors. To compile for the MCF52274 or MCF52277 processor, use `-mcpu=52274` or `-mcpu=52277`, respectively.

3.3.3. Changes in Sourcery G++ Lite 4.2-125

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

3.3.4. Changes in Sourcery G++ Lite 4.2-109

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

3.3.5. Changes in Sourcery G++ Lite 4.2-103

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.

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.

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

Disassembly of `%sr` and `%ccr` move instructions. Disassembly now correctly handles instructions that move a constant to the `%sr` or `%ccr` registers.

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.

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.

Instruction scheduling for ColdFire V4/V4e cores. GCC now supports instruction scheduling for ColdFire V4/V4e cores. This optimization is enabled by default with `-O2` or higher. You can disable the initial scheduling pass with `-fno-schedule-insns` and the final scheduling pass with `-fno-schedule-insns2`.

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.

Flash programming on m5485evb. The initialization sequence is corrected for the m5485evb board to permit flash programming. The erroneous sequence caused flash programming to fail.

Building large PIC object files. A new compiler option, `-mxgot`, has been added that removes a restriction on the number of external symbols that can be referenced in a PIC object file. If you see 'relocation truncated to fit: R_68K_GOT16O' errors from the linker, you should recompile the relevant object files with this option. However, since `-mxgot` carries both a speed and size penalty, you should use it only on files that otherwise result in link errors.

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.

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`*).

Errata fix for m52223 flash memory. The CS3 startup code for m52223 devices now implements a work-around for a hardware problem with speculative access to internal flash memory. As described in the Freescale errata MCF52223DE, work-around 1 is implemented.

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.

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.

3.3.6. Changes in Sourcery G++ Lite 4.2-47

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.

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.

Improved memcpy and memset. `memcpy` and `memset` have been optimized for better performance.

ColdFire ROM script error. A syntax error is fixed that prevented the ColdFire ROM scripts working for systems with ColdFire Flash Modules.

Internal compiler error bug fix. A bug that caused internal compiler errors when generating debug information in some cases has been fixed.

ColdFire 51QE support. CS3 now includes preliminary support for MCF51QE64 and MCF51QE128 boards.

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.

Board config file. The XML format for board configuration files used by the Sourcery G++ Debug Sprite has been changed to have more consistent syntax. Please consult Section 7.10, “Board File Syntax” for details of the XML.

Coprocessor instructions. Support has been added for ColdFire coprocessor 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.

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.

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

Single stepping with interrupts. A bug in the Debug Sprite has been fixed that previously caused failures in single stepping on some processors when there are active interrupt sources. The problem affected revision C and D debug hardware, which lack an Ignore Pending Interrupts bit, and caused the debugger to erroneously single step into interrupt handlers. The Sprite now raises the interrupt level when single stepping on such processors, in order to prevent entering interrupt handlers. Free running execution is not affected.

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.

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.

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.

Instruction scheduling for ColdFire architecture. GCC now supports instruction scheduling for ColdFire V1, V2 and V3 cores. This optimization is enabled by default with `-O2` or higher. You can disable the initial scheduling pass with `-fno-schedule-insns` and the final scheduling pass with `-fno-schedule-insns2`.

P&E driver for Linux. The P&E Linux driver has been updated to 3.24-811. The driver is required by the Sourcery G++ Debug Sprite to access P&E USB devices. Refer to Section 7.4.2, "Installing P&E Drivers" for installation instructions.

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.

ColdFire `wdebug.l` instructions. The assembler now correctly recognizes `wdebug.l` mnemonics. Previously only `wdebug` variants were allowed.

Turbo BDM Light ColdFire support. Support has been added for Turbo BDM Light ColdFire (TBLCF) devices, for example the Axiom AxBDM. You may now use these devices with the Debug Sprite to troubleshoot your ColdFire project. TBLCF support currently works on Windows hosts only.

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.

Debug Sprite on Windows. A defect that caused the Debug Sprite to hang at startup on some versions of Windows has been corrected.

P&E Windows driver update. The P&E Windows drivers have been updated to version 3.25.

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.

`-mno-strict-align` default. GCC now defaults to `-mno-strict-align` on ColdFire targets. This produces better code for unaligned memory accesses.

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.

Exception handling fix. A bug that caused the `a5` CPU register to contain incorrect values upon entry to exception handlers has been fixed.

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.

ColdFire 51QE processor support. Sourcery G++ now supports Freescale ColdFire 51QE processor. You can specify the `-mcpu=51qe` option to GCC and GAS to generate code for this processor.

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

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.

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.

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.

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

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.

3.3.7. Changes in Sourcery G++ Lite 4.2-8

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

3.3.8. Changes in Sourcery G++ Lite 4.2-4

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.

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.

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.

ColdFire ISA C support. Support is added for ColdFire ISA C code generation. The 54455 family of CPUs is now supported.

Execution order of `.fini_array` entries. For ELF targets using `.init_array` and `.fini_array` sections, the functions whose addresses are stored in `.fini_array` are now run in reverse order (i.e., the last function in the array is run first), as required by the ELF specification. Previous releases ran the entries in forward order (i.e., the first function in the array was run first).

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.

ColdFire RAMBAR control register. The ColdFire `RAMBAR` control register is numbered differently on different ColdFire devices. The assembler now accepts `RAMBAR` on all ColdFire devices and assigns it the appropriate device-specific number. The assembler also accepts the unambiguous `RAMBAR0` and `RAMBAR1` names. The disassembler always uses the unambiguous names.

3.3.9. Changes in Sourcery G++ Lite 4.2-2

GCC version 4.2. Sourcery G++ Lite for ColdFire ELF 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`.

Handling of out-of-range values by `strtof`. The `strtof` function now sets `errno` to `ERANGE` when the input is not representable as a `float`, as required by the ISO C standard.

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.

Control register access. Control registers can now be read from or written to in GDB or the Sourcery G++ IDE. In the IDE, they appear as structure members in the `Registers` pane.

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

Using internal SRAM. The linker scripts now define additional memory regions such as `.RAMBAR`. Objects can be placed in these regions by using the `__attribute__((section(".RAMBAR")))` attribute. Program start up initializes the contents of these regions.

Linker scripts. Board linker scripts are now only placed in the default multilib directory, or the multilib directory for the board in question. Therefore if you use an incorrect `-mcpu` option when compiling, you will get a link error rather than a program that will not run on your target board.

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.

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.

Assembler. The assembler is corrected to use the documented ColdFire control register names in addition to the Motorola M68K equivalents.

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

New board support. Support has been added for Cobra52235, Cobra5272, Cobra5282, Cobra5329, Cobra5475 and Cobra5485 boards.

New CPU support. Support for ColdFire 52221, 52223 and 5253 CPUs has been added.

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

3.3.10. Changes in Sourcery G++ Lite 4.1-39

Compiler. A bug is fixed where selecting multiple CPU types would result in a misleading error message.

3.3.11. 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.

3.3.12. Changes in Sourcery G++ Lite 4.1-36

Interrupt vector. The interrupt vector is now placed in a `.interrupt_vector` section and names unique functions so that any slot can be overridden easily.

Libraries for m5249 & m5250. Separate libraries are now built for m5249 and m5250 CPUs to avoid a linker warning.

Eclipse. Support is added for m5249 CPU and m5249c3 board. The Debug Sprite configure tabs are simplified. A bug preventing selection of a custom config file is fixed.

Linker Scripts. An incorrect additional “_” has been removed from the “_start” symbol. The default stack is set from the end of the “ram” memory region, rather than directly.

3.3.13. 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.

3.3.14. 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.

3.3.15. Changes in Sourcery G++ Lite 4.1-32

Interrupt functions. A bug is fixed that would cause functions defined with the `interrupt` attribute to generate an internal compiler error.

Debug Sprite. Additional CCS examples are provided. A confusing error message concerning the P&E library is clarified.

Floating-point registers. A bug is fixed that could cause functions using only floating-point registers to have an incorrect return sequence.

Wide characters. A bug in glibc is fixed that could affect copying of wide character strings.

3.3.16. 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.

3.3.17. Changes in Sourcery G++ Lite 4.1-30

Semihosting. Semihosting no longer uses trap 15. Thus all interrupt vectors are available for target program use.

Target features. The XML config file now contains an optional `<target-feature>` element to describe features of the target system. This can contain a `<floating-point>` element to specify that floating-point variables are available on the target CPU. The Debug Sprite's `-f` option is therefore deprecated.

P&E drivers. Drivers for P&E hardware are now included.

Command Converter Server. Command Converter Server (CCS) capability has been added to the Sourcery G++ Lite Debug Sprite.

Compiler. Additional `-mcpu` options are available to match the options accepted by the assembler. New preprocessor #defines (`__mcf_cpu_`*`NNNN`* and `__mcf_family_`*`NNNN`*) are created to specify which CPU is being targeted.

Startup. Program initialization now enables caching when available.

Linker scripts. Both hosted and unhosted linker scripts are now provided. The hosted scripts allow use of semihosting features. The unhosted scripts allow deployment without requiring an attached debugger.

Debugging Enhancements. It is now possible to set breakpoints on the first or last line of a function, even if there are no statements on that line.

3.3.18. 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.

3.3.19. 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.

3.3.20. 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.

3.3.21. 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.

3.3.22. Changes in Sourcery G++ Lite 4.1-20

Board Description. An XML style configuration file is available. This is used for both toolchain provided configs and user written configurations.

Debug Device. A URL style debug device specification is now used. This replaces several existing options with a uniform syntax that allows other ColdFire debug devices to be used in the future.

Compiler. A bug is fixed that caused an internal compiler error when compiling the Linux kernel at `-Os` optimization. The error is not specific to the Linux kernel and could affect other sources too.

3.3.23. 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.

3.3.24. Changes in Sourcery G++ Lite 4.1-17

Debug Sprite. A problem with single stepping when interrupts are active is fixed. The debugger no longer finds itself at the start of any interrupt routine after a single step. The instruction cache is now correctly invalidated when memory is updated. The debug sprite scans USB devices first, as these can provide confirmation that they are present. Other P&E device types may appear present even when no hardware is connected. The search order is reflected in the output of the `-i` option.

Newlib. A program's entry point is now called `_start` in line with other systems. `start` is provided as a weak alias, so that existing linker scripts still function.

Compiler. A bug is fixed where some floating-point instructions were not rounding their result to single precision. Usually this excess precision is harmless, but can effect some numerical programs.

3.3.25. 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.

3.3.26. Changes in Sourcery G++ Lite 4.1-14

uClinux improvements. There have been several changes to the uClinux toolchain:

- The toolchain and library now support position-independent code.
- The toolchain and library now support flat-format shared libraries.
- The toolchain now supports prioritized constructors and destructors.
- The linker can now reduce the size of flat objects by merging constant data and by optimizing unwind information.
- The `.gdb` file associated with each flat object now contains the original relocations; this is the same effect as you get by passing `--emit-relocs` to the linker. The temporary `.elf` work-around in version 4.1-12 has now been removed.

3.3.27. Changes in Sourcery G++ Lite 4.1-12

Semihosting support. Semihosting has been significantly improved when using a BDM interface to GDB. File system, time and remote program invocation is now provided over the GDB protocol. The newlib library provided with the m68k-elf version of the toolchain automatically uses semihosting to provide low level io facilities.

Support for Kirin2e & Kirin2u. Support for the Kirin2e & Kirin2u processors has been added. These are selectable with the `-mcpu=52235` and `-mcpu=5225` options respectively.

GDB improvements. Several improvements and bug fixes to debugging have been made. In particular:

- Hardware watchpoint support has been added.
- Hardware breakpoint support has been added.
- Several bug fixes relating to how function return values were determined.
- Speed improvements to the Sourcery G++ Lite Debug Stub. The number of requests between GDB and the debug stub, and between the debug stub and the target hardware have been reduced.

3.3.28. Changes in Sourcery G++ Lite 4.1-11

uClinux improvements. Several significant bug fixes in uClinux & uClibc. In particular:

- The stack could become misaligned leading to argument corruption.
- A fix for **elf2flt** so it works on Windows regardless of cygwin. A workaround to allow CodeWarrior to use the intermediate **.elf** files.

Support for m5329evb. Linker scripts and debug stub configuration has been added for this board.

3.3.29. 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 `m68k-elf`.

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 3.0 (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.

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.

4.2.3. Target System Requirements

See Chapter 3, *Sourcery G++ Lite for ColdFire ELF* 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.

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

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, or as a compressed archive `.tar.bz2`. If installing on a RPM-based GNU/Linux system you may download the `.rpm` file.

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.

4.4.1. Installing Sourcery G++ Lite 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
```

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 on Solaris or GNU/Linux Hosts from a Compressed Archive

You do not need to be a system administrator to install Sourcery G++ Lite on a GNU/Linux or Solaris system. You may install Sourcery G++ Lite using any user account and in any directory to which you have write access. This guide assumes that you have decided to install Sourcery G++ Lite in the `$HOME/CodeSourcery` subdirectory of your home directory and that the filename of the package you have downloaded is `/path/to/package.tar.bz2`. After installation the toolchain will be in `$HOME/CodeSourcery/sourceryg++-4.3`.

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
```

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

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

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

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

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

4.5. Installing Sourcery G++ Lite Updates

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

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

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

Note that the names of the Sourcery G++ commands for the ColdFire ELF target all begin with **m68k-elf**. 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 your host system. 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. Uninstalling Sourcery G++ Lite on Microsoft Windows

Select `Start`, then `Control Panel`. Select `Add or Remove Programs`. Scroll down and click on `Sourcery G++ for ColdFire ELF`. Select `Change/Remove` and 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` to remove the drivers separately. Depending on the device, you may need to reboot your computer to complete the driver uninstall.

4.6.2. Uninstalling Sourcery G++ Lite on Microsoft Windows Vista

Select `Start`, then `Settings` and finally `Control Panel`. Select the `Uninstall a program` task. Scroll down and double click on `Sourcery G++ for ColdFire ELF`. 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 `Uninstall a program` to remove the drivers separately. Depending on the device, you may need to reboot your computer to complete the driver uninstall.

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

If you installed Sourcery G++ Lite on GNU/Linux using the installer script, then you must use the corresponding uninstaller to remove Sourcery G++ Lite. The `m68k-elf` 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.

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

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

```
> rpm -e sourceryg++-m68k-elf
```

4.6.5. Uninstalling Sourcery G++ Lite on GNU/Linux or Solaris

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

On a non-Vista Microsoft Windows system, the installer automatically adds Sourcery G++ to your `PATH`. You can test that your `PATH` is set up correctly by using the following command:

```
> m68k-elf-g++ -v
```

and verifying that the last line of the output contains: `Sourcery G++ Lite 4.3-54`.

On a Microsoft Windows Vista system, the installer does not automatically add Sourcery G++ to your `PATH`. To set up your `PATH` on Microsoft Windows Vista, 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. You can verify that the command worked by starting a second `cmd.exe` shell and running:

```
> m68k-elf-g++ -v
```

Verify that the last line of the output contains: `Sourcery G++ Lite 4.3-54`.

4.7.1.1. 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 or Solaris

If you installed Sourcery G++ Lite using the `.bin` graphical installer then you may skip this step. The graphical 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 $HOME/CodeSourcery/sourceryg++-4.3/bin:$PATH
```

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

```
> PATH=$HOME/CodeSourcery/sourceryg++-4.3/bin:$PATH
> export PATH
```

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

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

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

```
> m68k-elf-g++
```

and verifying that you receive the message:

```
m68k-elf-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 m68k-elf, as indicated by the **m68k-elf** command prefix.

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

```
#include <stdio.h>

int
main (void)
{
  printf("Hello World!\n");
  return 0;
}
```

Compile and link this program using the command:

```
> m68k-elf-gcc -o hello hello.c -T script
```

Sourcery G++ requires that you specify a linker script with the `-T` option to build applications for bare-board targets. Linker errors like `undefined reference to `read'` are a symptom of failing to use an appropriate linker script. Default linker scripts are provided in `m68k-elf/lib`. Refer to Chapter 6, *CS3™: The CodeSourcery Common Startup Code Sequence* for information about the boards and linker scripts supported by Sourcery G++ Lite.

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

5.2. Running Applications on the Target System

Consult your target board documentation for instructions on loading programs onto the target, and running them. Alternatively, you can use the Sourcery G++ Debug Sprite from within GDB to download and run programs on the target via a supported hardware debugging device. See Chapter 7, *Sourcery G++ Debug Sprite* for details.

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

You can start the Sprite directly from within GDB:

```
(gdb) target remote | m68k-elf-sprite arguments
```

Refer to Section 7.3, “Sourcery G++ Debug Sprite Options” for a full description of the Sprite arguments.

5.3.2. Connecting to an External GDB Server

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

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

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

Chapter 6 CS3™: The CodeSourcery Common Startup Code Sequence

CS3 is CodeSourcery's low-level board support library. This chapter describes the organization of the system startup code and tells you how you can customize it, such as by defining your own interrupt handlers. This chapter also documents the boards supported by Sourcery G++ Lite and the compiler and linker options you need to use with them.

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

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

6.1. Startup Sequence

CS3 divides the startup sequence into three phases:

- In the *hard reset phase*, we initialize the memory controller and set up the memory map.
- In the *assembly initialization phase*, we prepare the stack to run C code, and jump to the C initialization function.
- In the *C initialization phase*, we initialize the data areas, run constructors for statically-allocated objects, and call `main`.

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

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

6.1.1. The Hard Reset Phase

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

The hard reset code is in a section named `.cs3.reset`. The section must define a symbol named `__cs3_reset_`*`sys`*, where *`sys`* is a name for the board being initialized; for example, the reset code for a m51qe128evb Eval (20MHz) board would be named `__cs3_reset_m51qe128evb_20`. The linker script defines the symbol `__cs3_reset` to refer to this reset address. If you need to refer to the reset address from generic code, you can use this non-specific `__cs3_reset` name.

When using the Sourcery G++ Debug Sprite, the Sprite is responsible for carrying out the initialization done in this phase. In this case execution begins at the start of the assembly initialization phase, except for simulators as described below.

Some simulators provide a supervisory operation to determine the amount of available memory. This operation is performed in the hard reset phase. Thus for simulators, execution always begins at `__cs3_reset_`*`sys`*.

The CodeSourcery board support library provides reasonable default reset code, but you may provide your own reset code by defining `__cs3_reset_`*`sys`* in an object file or library, in a `.cs3.reset` section.

6.1.2. The Assembly Initialization Phase

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

The value of the symbol `__cs3_stack` provides the initial value of the stack pointer. The CodeSourcery linker scripts provide a default value for this symbol, which you may override by defining `__cs3_stack` yourself.

Some processors initialize the stack pointer automatically on reset. However, because the assembly initialization phase is executed during debugging, it is required to set the stack pointer explicitly here in all cases.

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

Finally, we call the C function `__cs3_start_c`. This function never returns, and `_start` need not be prepared to handle a return from it.

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

The symbol `_start` lacks the `__cs3` prefix, because many debuggers and integrated development environments assume that name is used for this purpose.

6.1.3. The C Initialization Phase

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

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

In this function we take the following steps:

- Initialize all `.data`-like sections by copying their contents.
- Clear all `.bss`-like sections.
- Run constructors for statically-allocated objects, recorded using whatever conventions are usual for C++ on the target architecture.

  CS3 reserves priorities from 0 to 100 for use by initialization code. You can handle tasks like enabling interrupts, initializing coprocessors, pointing control registers at interrupt vectors, and so on by defining constructors with appropriate priorities.
- Call `main` as appropriate.
- Call `exit`, if it is available.

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

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

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

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

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

The constructor function may have an arbitrary name; `__cs3_init_args` is used only for illustrative purposes here.

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

6.2. Exit and Embedded Systems

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

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

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

6.3. Memory Layout

The header file `cs3.h` declares variables and types that describe the layout of memory on the system to C code. The variables are defined by the CS3 linker script or in the board support library.

The following variables describe the regions of memory to be initialized at startup:

```
/* The number of elements in __cs3_regions.  */
const size_t __cs3_region_num;

/* An untyped object, aligned on an eight-byte boundary.  */
typedef unsigned char __cs3_byte_align8
                      __attribute__ ((aligned (8)));

struct __cs3_region
{
  /* Flags for this region.  None defined yet.  */
  unsigned flags;

  __cs3_byte_align8 *init;  /* Region's initial contents.  */
  __cs3_byte_align8 *data;  /* Region's start address.  */

  /* These sizes are always a multiple of eight.  */
  size_t init_size;     /* Size of initial data.  */
  size_t zero_size;     /* Additional size to be zeroed.  */
};

/* An array of memory regions.  __cs3_regions[0] describes
   the region holding .data and .bss.  */
extern const struct __cs3_region __cs3_regions[];
```

The following variables describe the area of memory to be used for the dynamically-allocated heap:

```
/* The addresses of these objects are the start and end of
   free space for the heap, typically following .data and .bss.
   However, &__cs3_heap_end may be zero, meaning that we must
   determine the heap limit in some other way --- perhaps via a
   supervisory operation on a simulator, or simply by treating
   the stack pointer as the limit.  */
extern __cs3_byte_align8 __cs3_heap_start[];
extern __cs3_byte_align8 __cs3_heap_end[];

/* The end of free space for the heap, or zero if we haven't been
   able to determine it.  It usually points to __cs3_heap_end,
   but in some configurations, may be overridden by a supervisory
   call in the reset code.  */
extern void *__cs3_heap_limit;
```

For each region named *R*, `cs3.h` declares the following variables:

```
/* The start of the region.  */
extern unsigned char __cs3_region_start_R[]
                     __attribute__ ((aligned (8)));

/* The region's size, in bytes.  */
extern const size_t __cs3_region_size_R;
```

At the assembly level, the linker script also defines symbols with the same names and values.

If the region is initialized, then `cs3.h` also declares the following variables, corresponding to the region's element in `__cs3_regions`:

```
/* The data we initialize the region with.  */
extern const unsigned char __cs3_region_init_R[]
                           __attribute__ ((aligned (8)));

/* The size of the initialized portion of the region.  */
extern const size_t __cs3_region_init_size_R;

/* The size of the additional portion to be zeroed.  */
extern const size_t __cs3_region_zero_size_R;
```

Any of these identifiers may actually be defined as preprocessor macros that expand to expressions of the appropriate type and value.

Like the `struct __cs3_region` members, these regions are all aligned on eight-byte boundaries, and their sizes are multiples of eight bytes.

CS3 linker scripts place the contents of sections named `.cs3.region-head.`*`R`* at the start of each memory region. Note that CS3 itself may want to place items (like interrupt vector tables) at these locations; if there is a conflict, CS3 raises an error at link time.

6.4. Interrupt Vectors and Handlers

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

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

- Some processors fetch an address from an array indexed by the interrupt number, and jump to that address. We call these *address vector* processors; 51qe systems are a typical example.
- Others multiply the interrupt number by some constant factor, add a base address, and jump directly to that address. Here, the interrupt vector consists of blocks of code, so we call these *code vector* processors; PowerPC systems are a typical example.

On address vector processors, the CS3 library provides an array of pointers to interrupt handlers named `__cs3_interrupt_vector_`*`form`*, occupying a section named `.cs3.interrupt_vector`, where *`form`* identifies the particular processor variant the vector is appropriate for. If the processor supports more than one variety of interrupt vector (for example, a full-length form and a shortened form), then *`form`* identifies the variety as well. Each entry in the vector holds a reference to a symbol named `__cs3_isr_`*`int`*, where *`int`* is the customary name of that interrupt on the processor, or a number if there is no consistently used name. The library further provides a reasonable default definition for each `__cs3_isr_`*`int`* handler routine.

To override an individual handler, provide your own definition for the appropriate `__cs3_isr_`*`int`* symbol. The definition need not be placed in any particular object file section.

Interrupt handlers may be written in C using the `interrupt` attribute. For example, to override the `__csr_isr_access_error` handler, use the following definition:

```
void __attribute__ ((interrupt)) __csr_isr_access_error (void)
{
  ... custom handler code ...
}
```

To override the entire interrupt vector, you can define `__cs3_interrupt_vector_`*`form`*, placing the definition in a section named `.cs3.interrupt_vector`. The linker script reports an error if the `.cs3.interrupt_vector` section is empty, to ensure that the definition of `__cs3_interrupt_vector_`*`form`* occupies the proper section.

You may define the vector in C with an array of pointers using the `section` attribute to place it in the appropriate section. For example, to override the interrupt vector on a m51qe128evb Eval (20MHz) board, make the following definition:

```
typedef void handler(void);
handler *__attribute__((section (".cs3.interrupt_vector")))
  __cs3_interrupt_vector_coldfire[] =
{ ... };
```

On code vector processors, we follow the same conventions, with the following exceptions:

- In addition to being named `__cs3_isr_`*`int`*, each interrupt handler must also occupy a section named `.cs3.interrupt_`*`int`*. Naturally, each handler must fit within a single interrupt vector entry.
- Instead of providing a default definition for `__cs3_interrupt_vector_`*`form`* in the library, the linker script gathers the `.cs3.interrupt_`*`int`* sections together, in the proper order and on the necessary address boundaries, and defines the `__cs3_interrupt_vector_`*`form`* symbol to refer to its start.

To override an individual handler on a code vector processor, you provide your own definition for `__cs3_isr_`*`int`*, placed in an appropriate section. The linker script ensures that each `.cs3.interrupt_`*`int`* section is non-empty, so that placing a handler in the wrong section elicits an error at link time.

CS3 does not allow you to override the entire interrupt vector on code vector processors, because the code vector must be constructed by the linker script, and thus cannot come from a library or object file. However, the portion of the linker script that constructs the interrupt vector occupies its own file, which other linker scripts can incorporate using the **INCLUDE** linker script command, making it easier to replace the linker script entirely and still take advantage of CS3's other features.

Some processors, like the Innovasic fido, use more than one interrupt vector: the processor provides several interrupt vector pointer registers, each used in different circumstances. Each register may point to a different vector, or some or all may share vectors.

On these processors, CS3 provides only a single pre-constructed interrupt vector, but defines a separate symbol for each interrupt vector pointer register; all the symbols point to the pre-constructed vector by default. The CS3 startup code initializes each register from the corresponding symbol. You can provide your own vectors by defining the appropriate symbols.

For example, the fido processor has five contexts, each of which can use its own interrupt vector; on this architecture, CS3 defines the standard `__cs3_interrupt_vector_fido` symbol referring to the pre-constructed vector, and then goes on to define per-context symbols `__cs3_interrupt_vector_fido_ctx0`, `__cs3_interrupt_vector_fido_ctx1`, and so on, all referring to `__cs3_interrupt_vector_fido`. The CS3 startup code sets each context's vector register to the value of the corresponding symbol. By default, all the contexts share an interrupt vector, but if your code provides its own definition for `__cs3_interrupt_vector_fido_ctx1`, then the startup code initializes context one's register to point to that vector instead.

This arrangement requires you to use a different approach to specify a handler for a secondary context that differs from the corresponding handler in the primary context. For example, to handle division-by-zero exceptions in context 1 with the function `ctx1_divide_by_zero`, you should write the following:

```
typedef void (*handler_type) (void);
handler_type __cs3_interrupt_vector_fido_ctx1[256];
extern handler_type __cs3_interrupt_vector_fido[256];
```

```
__attribute__((interrupt))
void
ctx1_divide_by_zero (void)
{
  /* Your code here.  */
}

__attribute__((constructor))
void
initialize_vector_ctx1 (void)
{
  /* Initialize our custom vector from the
     pre-constructed CS3 vector.  */
  memcpy (__cs3_interrupt_vector_fido_ctx1,
          __cs3_interrupt_vector_fido,
          sizeof (__cs3_interrupt_vector_fido));

  /* Initialize custom interrupt handlers.  */
  __cs3_interrupt_vector_fido_ctx1[5] = ctx1_divide_by_zero;
}
```

With this code in place, when a division-by-zero exception occurs in context 1, the processor calls `ctx1_divide_by_zero` to handle it. Defining `initialize_vector_ctx1` with the `constructor` attribute arranges for CS3 to call it before calling your program's `main` function.

6.5. Linker Scripts

CS3 provides linker scripts for each board that is supported. Each board may be used in a number of different configurations, and these are reflected in the linker script name. The linker scripts are named *`board-profile-hosted`*`.ld`, where *`board`* is the name of the board, *`profile`* describes the memory arrangement used and *`-hosted`* indicates whether semihosting is provided.

Caution

If you do not explicitly specify a linker script, Sourcery G++ produces a link error, preventing you from creating an executable program.

6.5.1. Program and Data Placement

Many boards have both RAM and ROM (flash) memory devices. CS3 provides distinct linker scripts to place the application either entirely in RAM, or in ROM where data is initialized during the C initialization phase.

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

6.5.2. Semihosting

CS3 is designed to support boards where there may be no operating system. To allow functions like `open` and `write` to work, a *semihosting* feature is supported, in conjunction with the debugger.

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

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

```
#include <unistd.h>

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

The hosted CS3 linker scripts provide the semihosting support, and as such programs linked with them may only be run with the debugger. The unhosted CS3 linker scripts provide stub versions of the system calls, which return an appropriate error value in `errno`. If such a stub system call is required in the executable, the linker also produces a warning. Such a warning may indicate that you have left debugging code active, and that your executable is larger than it might need to be.

6.5.3. Choosing a Linker Script from the Command Line

From the command line, you must add `-T` *`script`* to your linking command, where *`script`* is the appropriate linker script. For example, if you are using a m51qe128evb Eval (20MHz) board, you could link with `-T m51qe128evb-20-ram-hosted.ld`.

6.6. Supported Boards for ColdFire ELF

The following linker scripts are provided for m68k-elf.

6.6.1. m51qe128evb Eval (20MHz)

- Processor name: 51qe
- Processor options: `-mcpu=51qe`
- Memory regions: `ram`, `rom`
- Other regions: `gpio`, `io`

Profile & Hosting	**Linker Script**	**Reset name**
RAM Hosted	`m51qe128evb-20-ram-hosted.ld`	`__cs3_reset_m51qe128evb_20`
RAM Unhosted	`m51qe128evb-20-ram.ld`	`__cs3_reset_m51qe128evb_20`
ROM Hosted	`m51qe128evb-20-rom-hosted.ld`	`__cs3_reset_m51qe128evb_20`
ROM Unhosted	`m51qe128evb-20-rom.ld`	`__cs3_reset_m51qe128evb_20`

6.6.2. m51qe128evb Eval (40MHz)

- Processor name: 51qe

- Processor options: `-mcpu=51qe`
- Memory regions: `ram`, `rom`
- Other regions: `gpio`, `io`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m51qe128evb-40-ram-hosted.ld`	`__cs3_reset_m51qe128evb_40`
RAM Unhosted	`m51qe128evb-40-ram.ld`	`__cs3_reset_m51qe128evb_40`
ROM Hosted	`m51qe128evb-40-rom-hosted.ld`	`__cs3_reset_m51qe128evb_40`
ROM Unhosted	`m51qe128evb-40-rom.ld`	`__cs3_reset_m51qe128evb_40`

6.6.3. m51qe128evb Eval (50.33MHz)

- Processor name: 51qe
- Processor options: `-mcpu=51qe`
- Memory regions: `ram`, `rom`
- Other regions: `gpio`, `io`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m51qe128evb-50-ram-hosted.ld`	`__cs3_reset_m51qe128evb_50`
RAM Unhosted	`m51qe128evb-50-ram.ld`	`__cs3_reset_m51qe128evb_50`
ROM Hosted	`m51qe128evb-50-rom-hosted.ld`	`__cs3_reset_m51qe128evb_50`
ROM Unhosted	`m51qe128evb-50-rom.ld`	`__cs3_reset_m51qe128evb_50`

6.6.4. m51qe32evb Eval (20MHz)

- Processor name: 51qe
- Processor options: `-mcpu=51qe`
- Memory regions: `ram`, `rom`
- Other regions: `gpio`, `io`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m51qe32evb-20-ram-hosted.ld`	`__cs3_reset_m51qe32evb_20`
RAM Unhosted	`m51qe32evb-20-ram.ld`	`__cs3_reset_m51qe32evb_20`
ROM Hosted	`m51qe32evb-20-rom-hosted.ld`	`__cs3_reset_m51qe32evb_20`

Profile & Hosting	Linker Script	Reset name
ROM Unhosted	`m51qe32evb-20-rom.ld`	`__cs3_reset_m51qe32evb_20`

6.6.5. m51qe32evb Eval (40MHz)

- Processor name: 51qe
- Processor options: `-mcpu=51qe`
- Memory regions: `ram`, `rom`
- Other regions: `gpio`, `io`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m51qe32evb-40-ram-hosted.ld`	`__cs3_reset_m51qe32evb_40`
RAM Unhosted	`m51qe32evb-40-ram.ld`	`__cs3_reset_m51qe32evb_40`
ROM Hosted	`m51qe32evb-40-rom-hosted.ld`	`__cs3_reset_m51qe32evb_40`
ROM Unhosted	`m51qe32evb-40-rom.ld`	`__cs3_reset_m51qe32evb_40`

6.6.6. m51qe32evb Eval (50.33MHz)

- Processor name: 51qe
- Processor options: `-mcpu=51qe`
- Memory regions: `ram`, `rom`
- Other regions: `gpio`, `io`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m51qe32evb-50-ram-hosted.ld`	`__cs3_reset_m51qe32evb_50`
RAM Unhosted	`m51qe32evb-50-ram.ld`	`__cs3_reset_m51qe32evb_50`
ROM Hosted	`m51qe32evb-50-rom-hosted.ld`	`__cs3_reset_m51qe32evb_50`
ROM Unhosted	`m51qe32evb-50-rom.ld`	`__cs3_reset_m51qe32evb_50`

6.6.7. m51qe64evb Eval (20MHz)

- Processor name: 51qe
- Processor options: `-mcpu=51qe`
- Memory regions: `ram`, `rom`
- Other regions: `gpio`, `io`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m51qe64evb-20-ram-hosted.ld`	`__cs3_reset_m51qe64evb_20`
RAM Unhosted	`m51qe64evb-20-ram.ld`	`__cs3_reset_m51qe64evb_20`
ROM Hosted	`m51qe64evb-20-rom-hosted.ld`	`__cs3_reset_m51qe64evb_20`
ROM Unhosted	`m51qe64evb-20-rom.ld`	`__cs3_reset_m51qe64evb_20`

6.6.8. m51qe64evb Eval (40MHz)

- Processor name: 51qe
- Processor options: `-mcpu=51qe`
- Memory regions: `ram`, `rom`
- Other regions: `gpio`, `io`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m51qe64evb-40-ram-hosted.ld`	`__cs3_reset_m51qe64evb_40`
RAM Unhosted	`m51qe64evb-40-ram.ld`	`__cs3_reset_m51qe64evb_40`
ROM Hosted	`m51qe64evb-40-rom-hosted.ld`	`__cs3_reset_m51qe64evb_40`
ROM Unhosted	`m51qe64evb-40-rom.ld`	`__cs3_reset_m51qe64evb_40`

6.6.9. m51qe64evb Eval (50.33MHz)

- Processor name: 51qe
- Processor options: `-mcpu=51qe`
- Memory regions: `ram`, `rom`
- Other regions: `gpio`, `io`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m51qe64evb-50-ram-hosted.ld`	`__cs3_reset_m51qe64evb_50`
RAM Unhosted	`m51qe64evb-50-ram.ld`	`__cs3_reset_m51qe64evb_50`
ROM Hosted	`m51qe64evb-50-rom-hosted.ld`	`__cs3_reset_m51qe64evb_50`
ROM Unhosted	`m51qe64evb-50-rom.ld`	`__cs3_reset_m51qe64evb_50`

6.6.10. m5206ec3 Eval

- Processor name: 5206e
- Processor options: `-mcpu=5206e`
- Memory regions: `ram`, `rom`, `rambar`
- Other regions: `mbar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5206ec3-ram-hosted.ld`	`__cs3_reset_m5206ec3`
RAM Unhosted	`m5206ec3-ram.ld`	`__cs3_reset_m5206ec3`
ROM Hosted	`m5206ec3-rom-hosted.ld`	`__cs3_reset_m5206ec3`
ROM Unhosted	`m5206ec3-rom.ld`	`__cs3_reset_m5206ec3`

6.6.11. m5208evb Eval

- Processor name: 5208 (MiniMi)
- Processor options: `-mcpu=5208`
- Memory regions: `ram`, `rom`, `rambar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5208evb-ram-hosted.ld`	`__cs3_reset_m5208evb`
RAM Unhosted	`m5208evb-ram.ld`	`__cs3_reset_m5208evb`
ROM Hosted	`m5208evb-rom-hosted.ld`	`__cs3_reset_m5208evb`
ROM Unhosted	`m5208evb-rom.ld`	`__cs3_reset_m5208evb`

6.6.12. m5208evb RevE Eval

- Processor name: 5208 (MiniMi)
- Processor options: `-mcpu=5208`
- Memory regions: `ram`, `rom`, `rambar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5208evbe-ram-hosted.ld`	`__cs3_reset_m5208evbe`
RAM Unhosted	`m5208evbe-ram.ld`	`__cs3_reset_m5208evbe`
ROM Hosted	`m5208evbe-rom-hosted.ld`	`__cs3_reset_m5208evbe`
ROM Unhosted	`m5208evbe-rom.ld`	`__cs3_reset_m5208evbe`

6.6.13. m5213evb Eval (66MHz)

- Processor name: 5213 (Kirin)
- Processor options: `-mcpu=5213`
- Memory regions: `ram`, `rom`
- Other regions: `ipsbar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5213evb-66-ram-hosted.ld`	`__cs3_reset_m5213evb_66`
RAM Unhosted	`m5213evb-66-ram.ld`	`__cs3_reset_m5213evb_66`
ROM Hosted	`m5213evb-66-rom-hosted.ld`	`__cs3_reset_m5213evb_66`
ROM Unhosted	`m5213evb-66-rom.ld`	`__cs3_reset_m5213evb_66`

6.6.14. m5213evb Eval (80MHz)

- Processor name: 5213 (Kirin)
- Processor options: `-mcpu=5213`

- Memory regions: `ram`, `rom`
- Other regions: `ipsbar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5213evb-80-ram-hosted.ld`	`__cs3_reset_m5213evb_80`
RAM Unhosted	`m5213evb-80-ram.ld`	`__cs3_reset_m5213evb_80`
ROM Hosted	`m5213evb-80-rom-hosted.ld`	`__cs3_reset_m5213evb_80`
ROM Unhosted	`m5213evb-80-rom.ld`	`__cs3_reset_m5213evb_80`

6.6.15. m52223evb Eval (66MHz)

- Processor name: 52223
- Processor options: `-mcpu=52223`
- Memory regions: `ram`, `rom`
- Other regions: `ipsbar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m52223evb-66-ram-hosted.ld`	`__cs3_reset_m52223evb_66`
RAM Unhosted	`m52223evb-66-ram.ld`	`__cs3_reset_m52223evb_66`
ROM Hosted	`m52223evb-66-rom-hosted.ld`	`__cs3_reset_m52223evb_66`
ROM Unhosted	`m52223evb-66-rom.ld`	`__cs3_reset_m52223evb_66`

6.6.16. m52223evb Eval (80MHz)

- Processor name: 52223
- Processor options: `-mcpu=52223`
- Memory regions: `ram`, `rom`
- Other regions: `ipsbar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m52223evb-80-ram-hosted.ld`	`__cs3_reset_m52223evb_80`
RAM Unhosted	`m52223evb-80-ram.ld`	`__cs3_reset_m52223evb_80`
ROM Hosted	`m52223evb-80-rom-hosted.ld`	`__cs3_reset_m52223evb_80`
ROM Unhosted	`m52223evb-80-rom.ld`	`__cs3_reset_m52223evb_80`

6.6.17. m52235evb Eval (60MHz)

- Processor name: 52235
- Processor options: `-mcpu=52235`
- Memory regions: `ram`, `rom`
- Other regions: `ipsbar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m52235evb-ram-hosted.ld`	`__cs3_reset_m52235evb`
RAM Unhosted	`m52235evb-ram.ld`	`__cs3_reset_m52235evb`
ROM Hosted	`m52235evb-rom-hosted.ld`	`__cs3_reset_m52235evb`
ROM Unhosted	`m52235evb-rom.ld`	`__cs3_reset_m52235evb`

6.6.18. m5235evb Eval

- Processor name: 5235
- Processor options: `-mcpu=5235`
- Memory regions: `ram`, `rom`
- Other regions: `ipsbar`, `rambar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5235evb-ram-hosted.ld`	`__cs3_reset_m5235evb`
RAM Unhosted	`m5235evb-ram.ld`	`__cs3_reset_m5235evb`
ROM Hosted	`m5235evb-rom-hosted.ld`	`__cs3_reset_m5235evb`
ROM Unhosted	`m5235evb-rom.ld`	`__cs3_reset_m5235evb`

6.6.19. m5249c3 Eval

- Processor name: 5249
- Processor options: `-mcpu=5249`
- Memory regions: `ram`, `rom`, `rambar0`, `rambar1`
- Other regions: `mbar`, `mbar2`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5249c3-ram-hosted.ld`	`__cs3_reset_m5249c3`
RAM Unhosted	`m5249c3-ram.ld`	`__cs3_reset_m5249c3`
ROM Hosted	`m5249c3-rom-hosted.ld`	`__cs3_reset_m5249c3`
ROM Unhosted	`m5249c3-rom.ld`	`__cs3_reset_m5249c3`

6.6.20. m5253evb Eval

- Processor name: 5253 (Amadeus)
- Processor options: `-mcpu=5253`
- Memory regions: `ram`, `rom`, `rambar0`, `rambar1`
- Other regions: `mbar`, `mbar2`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5253evb-ram-hosted.ld`	`__cs3_reset_m5253evb`

Profile & Hosting	Linker Script	Reset name
RAM Unhosted	`m5253evb-ram.ld`	`__cs3_reset_m5253evb`
ROM Hosted	`m5253evb-rom-hosted.ld`	`__cs3_reset_m5253evb`
ROM Unhosted	`m5253evb-rom.ld`	`__cs3_reset_m5253evb`

6.6.21. m5272c3 Eval

- Processor name: 5272
- Processor options: `-mcpu=5272`
- Memory regions: `ram`, `rom`, `rambar`
- Other regions: `mbar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5272c3-ram-hosted.ld`	`__cs3_reset_m5272c3`
RAM Unhosted	`m5272c3-ram.ld`	`__cs3_reset_m5272c3`
ROM Hosted	`m5272c3-rom-hosted.ld`	`__cs3_reset_m5272c3`
ROM Unhosted	`m5272c3-rom.ld`	`__cs3_reset_m5272c3`

6.6.22. m5275evb Eval

- Processor name: 5275
- Processor options: `-mcpu=5275`
- Memory regions: `ram`, `rom`, `rambar`
- Other regions: `ipsbar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5275evb-ram-hosted.ld`	`__cs3_reset_m5275evb`
RAM Unhosted	`m5275evb-ram.ld`	`__cs3_reset_m5275evb`
ROM Hosted	`m5275evb-rom-hosted.ld`	`__cs3_reset_m5275evb`
ROM Unhosted	`m5275evb-rom.ld`	`__cs3_reset_m5275evb`

6.6.23. m5282evb Eval

- Processor name: 5282
- Processor options: `-mcpu=5282`
- Memory regions: `ram`, `rom`, `rombar`, `rambar`
- Other regions: `ipsbar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5282evb-ram-hosted.ld`	`__cs3_reset_m5282evb`
RAM Unhosted	`m5282evb-ram.ld`	`__cs3_reset_m5282evb`

Profile & Hosting	Linker Script	Reset name
ROM Hosted	`m5282evb-rom-hosted.ld`	`__cs3_reset_m5282evb`
ROM Unhosted	`m5282evb-rom.ld`	`__cs3_reset_m5282evb`

6.6.24. m5307c3 Eval

- Processor name: 5307
- Processor options: `-mcpu=5307`
- Memory regions: `ram`, `rom`, `rambar`, `sram`
- Other regions: `mbar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5307c3-ram-hosted.ld`	`__cs3_reset_m5307c3`
RAM Unhosted	`m5307c3-ram.ld`	`__cs3_reset_m5307c3`
ROM Hosted	`m5307c3-rom-hosted.ld`	`__cs3_reset_m5307c3`
ROM Unhosted	`m5307c3-rom.ld`	`__cs3_reset_m5307c3`

6.6.25. m5329evb Eval

- Processor name: 5329 (DragonFire)
- Processor options: `-mcpu=5329`
- Memory regions: `ram`, `rom`, `rambar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m5329evb-ram-hosted.ld`	`__cs3_reset_m5329evb`
RAM Unhosted	`m5329evb-ram.ld`	`__cs3_reset_m5329evb`
ROM Hosted	`m5329evb-rom-hosted.ld`	`__cs3_reset_m5329evb`
ROM Unhosted	`m5329evb-rom.ld`	`__cs3_reset_m5329evb`

6.6.26. m54455evb Eval

- Processor name: 54455 (RedStripe)
- Processor options: `-mcpu=54455`
- Memory regions: `ram`, `rom`, `rambar1`, `bootrom`
- Other regions: `mbar`

Profile & Hosting	Linker Script	Reset name
RAM Hosted	`m54455evb-ram-hosted.ld`	`__cs3_reset_m54455evb`
RAM Unhosted	`m54455evb-ram.ld`	`__cs3_reset_m54455evb`
ROM Hosted	`m54455evb-rom-hosted.ld`	`__cs3_reset_m54455evb`
ROM Unhosted	`m54455evb-rom.ld`	`__cs3_reset_m54455evb`

6.6.27. m5485evb Eval

- Processor name: 5485 (Rigoletto)
- Processor options: `-mcpu=5485`
- Memory regions: `ram`, `rom`, `rambar0`, `rambar1`, `bootrom`
- Other regions: `mbar`

Profile & Hosting	**Linker Script**	**Reset name**
RAM Hosted	`m5485evb-ram-hosted.ld`	`__cs3_reset_m5485evb`
RAM Unhosted	`m5485evb-ram.ld`	`__cs3_reset_m5485evb`
ROM Hosted	`m5485evb-rom-hosted.ld`	`__cs3_reset_m5485evb`
ROM Unhosted	`m5485evb-rom.ld`	`__cs3_reset_m5485evb`

6.7. Interrupt Vector Tables

The ColdFire interrupt vector table (`__cs3_interrupt_vector_coldfire`) contents are:

Number	**Name**	**Meaning**
0	`__cs3_stack`	Initial stack pointer
1	`__cs3_reset`	Reset entry point
2	`__cs3_isr_access_error`	Access error
3	`__cs3_isr_address_error`	Address error
4	`__cs3_isr_illegal_instruction`	
5	`__cs3_isr_divide_by_zero`	Divide by zero
6..7	`__cs3_isr_interrupt_6..7`	Unspecified
8	`__cs3_isr_privilege_violation`	Privilege violation
9	`__cs3_isr_trace`	Trace
10	`__cs3_isr_unimplemented_line_a_opcode`	Unimplemented instruction
11	`__cs3_isr_unimplemented_line_f_opcode`	
12	`__cs3_isr_non_pc_breakpoint_debug_interrupt`	Breakpoint
13	`__cs3_isr_pc_breakpoint_debug_interrupt`	
14	`__cs3_isr_format_error`	
15..23	`__cs3_isr_interrupt_15..23`	Unspecified
24	`__cs3_isr_spurious_interrupt`	
25..31	`__cs3_isr_interrupt_25..31`	Unspecified
32..47	`__cs3_isr_trap0..trap15`	User trap
48	`__cs3_isr_fp_branch_unordered`	Floating point error
49	`__cs3_isr_fp_inexact_result`	
50	`__cs3_isr_fp_divide_by_zero`	
51	`__cs3_isr_fp_underflow`	

Number	Name	Meaning
52	`__cs3_isr_fp_operand_error`	
53	`__cs3_isr_fp_overflow`	
54	`__cs3_isr_fp_input_not_a_number`	
55	`__cs3_isr_fp_input_denormalized_number`	
56..60	`__cs3_isr_interrupt_56..60`	Unspecified
61	`__cs3_isr_unsupported_instruction`	
62..255	`__cs3_isr_interrupt_62..255`	Unspecified

6.8. Regions and Memory Sections

The following regions are defined for ColdFire ELF.

Region	Contents
`bootrom`	External bootstrap flash
`gpio`	IO space
`io`	IO space
`ipsbar`	Internal memory mapped IO
`mbar`	Internal memory mapped IO
`mbar2`	Internal memory mapped IO
`ram`	`.data` and `.bss` sections. In ram-based profiles, also contains `.text` and other program-like sections.
`rambar`	Internal RAM
`rambar0`	Internal RAM
`rambar1`	Internal RAM
`rom`	`.text` and other program-like sections. Initialization values for data-like regions.
`rombar`	Internal flash
`sram`	External RAM

Note that not all regions are provided in every linker script or profile; see the documentation of the individual linker scripts in Section 6.6, “Supported Boards for ColdFire ELF”, above.

Regions documented as "Memory regions" correspond to similarly-named program sections. For example, the linker script assigns the `.ram` section to the `ram` region. You can explicitly locate data or code in these sections using section attributes in your source C or C++ code. Section attributes are especially useful on code compiled for boards that include special memory banks, such as a fast on-chip cache memory, in addition to the default `ram` and/or `rom` regions. CS3 arranges for additional data-like sections to be initialized in the same way as the default `.data` section.

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

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

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

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

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

Regions documented as "Other regions" do not have a corresponding program section. Typically, these regions correspond to memory-mapped control and I/O registers that cannot be used for general data or program storage. If you need to manipulate data in these regions, you can use the CS3 memory layout facilities declared in `cs3.h`, as described in Section 6.3, “Memory Layout”.

Memory maps for boards supported by Sourcery G++ Lite for ColdFire ELF are documented in the linker scripts in the `m68k-elf/lib/` subdirectory of your Sourcery G++ installation directory. Note that the memory maps are consistent with those used by Freescale's CodeWarrior tools.

Chapter 7
Sourcery G++ Debug Sprite

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

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

The Sprite acts as an interface between GDB and external debug devices and libraries. Refer to Section 7.2, "Invoking Sourcery G++ Debug Sprite" for information about the specific devices supported by this version of Sourcery G++ Lite.

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.

7.1. Debug Sprite Example

This section demonstrates execution and debugging of a simple application. Start by creating a file named `fib.c`:

```
#include <unistd.h>

static int Fib (unsigned n, unsigned a, unsigned b)
{
  unsigned count;

  for (count = 0; count != b; count++)
    write (1, ".", 1);
  write (1, "\n", 1);

  if (n)
    Fib (n - 1, b, a + b);
}

int main ()
{
  write (1, "Fibonacci\n", 10);
  Fib (9, 0, 1);
  return 0;
}
```

First compile and link the program for the target board. If it is a stand-alone program for a m51qe128evb Eval (20MHz) board use:

```
> m68k-elf-gcc -mcpu=51qe \
  -Tm51qe128evb-20-ram-hosted.ld fib.c -o fib -g
```

For other boards you must make appropriate substitutions in the preceding command. If your program is an operating system kernel such as uClinux or Linux, your usual build method should be adequate, as the kernel contains the necessary initialization code for interrupt handlers.

Verify that the Sourcery G++ Debug Sprite can detect your debug hardware:

```
> m68k-elf-sprite -i
```

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

```
CodeSourcery ColdFire Debug Sprite
    (Sourcery G++ Lite Sourcery G++ Lite 4.3-54)
pe: [speed=<n:0-31>&memory-timeout=<n:0-99>] P&E Adaptor
  pe://USBMultilink/PE6011970 - USB1 : USB-ML-CF Rev C (PE6011970)
  pe://USBMultilink/PE6011981 - USB2 : USB-ML-CF Rev C (PE6011981)
  pe://USBMultilink/PE6016152 - USB3 : USB-ML-CF Rev C (PE6016152)
  pe://CycloneProMaxEthernet/10.0.0.85 - 10.0.0.85  :  cyclone1
  pe://CycloneProMaxEthernet/10.0.0.86 - 10.0.0.86  :  cyclone2
ccs: [timeout=<n>&speed=<n>] CCS Adaptor
  ccs://$Host:$Port/$Chain_position - CCS address
tblcf: TBLCF Interface
  tblcf://0/ - TBLCF device
```

This shows that P&E devices, Command Converter Server and Turbo BDM Light ColdFire are supported. Five P&E devices are detected, and one TBLCF device. Command Converter Server (CCS) devices are supported, but cannot be autodetected.

Start the debugger on your host system:

```
> m68k-elf-gdb fib
```

Connecting GDB to the board depends on the debug device you are using. If you are using a P&E debug device, use:

```
(gdb) target remote | m68k-elf-sprite pe: m51qe128evb-20
Remote debugging using | m68k-elf-sprite pe: m51qe128evb-20
m68k-elf-sprite:Opening P&E USBMultilink port 1
    (USB1 : USB-ML-CF Rev C (PE6011970))
m68k-elf-sprite:Target reset
0x00008936 in ?? ()
```

See the additional information below for more details about using the Debug Sprite with P&E devices, or for instructions if you are using one of the other supported devices.

At this point you can use GDB to load your program onto the target board and control its execution as required:

```
(gdb) load
Loading section .text, size 0x2cf6 lma 0x20000000
Loading section .data, size 0x814 lma 0x20002cf8
Loading section .jcr, size 0x4 lma 0x2000350c
Start address 0x20000400, load size 13582
Transfer rate: 7197 bits/sec, 186 bytes/write.
```

Set a breakpoint so that the debugger stops when your program reaches `main`:

```
(gdb) break main
Breakpoint 1 at 0x20000524: file fib.c, line 17.
```

Allow the program to execute until it reaches `main`:

```
(gdb) continue
Continuing.
```

```
main () at fib.c:13
13         write (1, "Fibonacci\n", 10);
(gdb) next
Fibonacci
14         Fib (9, 0, 1);
```

Permit the program to finish executing with:

```
(gdb) continue
Continuing.
.
.
..
...
.....
........
.............
.....................
..................................
.......................................................

Program exited normally.
```

7.2. Invoking Sourcery G++ Debug Sprite

The Debug Sprite is invoked as follows:

```
m68k-elf-sprite [options] device-url board-file
```

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

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

Most device URL schemes also follow the regular format:

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

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

`pe`	Use a P&E Microcomputer Systems debugging device. Refer to Section 7.4, “P&E Devices”.
`ccs`	Use a debugging device controlled by the Command Converter Server (CCS) utility, such as a CodeWarrior Ethernet TAP or USB TAP. Refer to Section 7.5, “Command Converter Server Devices”.
`tblcf`	Use a Turbo BDM Light ColdFire (e.g. Axiom AxBDM) debugging device. Refer to Section 7.6, “Turbo BDM Light ColdFire 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 7.9, “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 7.10, “Board File Syntax” for more information.

7.3. Sourcery G++ Debug Sprite Options

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

`-b`	Print a list of *`board-file`* files in the board config directory.
`-h`	Print a list of options and their meanings. A list of *`device-url`* syntaxes is also shown.
`-i`	Print a list of the accessible devices. If a *`device-url`* is also specified, only devices for that device type are scanned. Each supported device type is listed along with the options that can be appended to the *`device-url`*. For each discovered device, the *`device-url`* is printed along with a description of that device.
`-l` *`[host]:port`*	Specify the host address and port number to listen for a GDB connection. If this option is not given, the Debug Sprite communicates with GDB using stdin and stdout. If you start the Sprite from within GDB using the `target remote | m68k-elf-sprite ...` command, you do not need this option.
`-m`	Listen for multiple sequential connections. Normally the Debug Sprite terminates after the first connection from GDB terminates. This option instead makes it listen for a subsequent connection. To terminate the Sprite, open a connection and send the string `END\n`.
`-q`	Do not print any messages.
`-v`	Print additional messages.

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

7.4. P&E Devices

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

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

The various parts are:

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

- `USBMultilink`
- `CycloneProMaxUSB`
- `CycloneProMaxSerial`

- `CycloneProMaxEthernet`
- `ParallelPortCable`
- `PCIBDMLightning`

`number`	Specify the debug device number. Be aware that a device's number depends on whether other devices are concurrently accessed (this is a feature of the underlying P&E library).
`key`	Some P&E devices report unique device keys. This option allows you to select a device by its key, independently of USB device numbering.

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

`speed=`*`speed`*	Specify the speed of the connection. This is a clock divider value, so higher values are slower connection speeds. Refer to the P&E documentation for valid speed settings for your board.
`memory-timeout=`*`timeout`*	Some boards report memory errors for every access within a certain time of a genuine memory error. This option instructs the Sprite to compensate for this and retry a memory access that reports an error within the specified time of a prior error. If you need to use this option you need to increase GDB's protocol timeout by specifying `set remotetimeout` *`N`* at the GDB prompt.
`debug=`*`file`*	Write P&E debug information to *`file`*.

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

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

7.4.1. Connection Problems

If you get a message `Cannot load P&E library 'UNIT_CFZ.DLL'` or `Cannot load P&E library 'libUnit_cfz.so'`, you probably have not installed the P&E device software. This software is included with Sourcery G++ Lite; see Section 7.4.2, “Installing P&E Drivers” for installation instructions.

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

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

7.4.2. Installing P&E Drivers

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

To reinstall the drivers on Windows, follow these steps:

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

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

These drivers are provided by P&E Microcomputer Systems.

7.5. Command Converter Server Devices

The Sourcery G++ Debug Sprite supports devices such as the CodeWarrior Ethernet TAP and USB TAP that are controlled by the Command Converter Server (CCS) utility. You need to start CCS separately before connecting to the debug device from GDB; see Section 7.5.1, “Starting CCS”.

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

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

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

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

`speed=`*`speed`*	Specify the speed used to connect to the target. This is specified in KHz by default. You can use `MHz` and `KHz` suffixes.
`timeout=`*`timeout`*	This specifies the timeout, in seconds, used for communication with the Command Converter Server.

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

```
(gdb) target remote | \
  m68k-elf-sprite ccs://localhost:41475 m51qe128evb-20
Remote debugging using |
  m68k-elf-sprite ccs://localhost:41475 m51qe128evb-20
m68k-elf-sprite:Connected to CCS version: 4.13
    CC version: 1:3
m68k-elf-sprite:Target reset
0x00008936 in ?? ()
```

7.5.1. Starting CCS

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

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

Use the following commands to initialize the server:

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

The *`port`* number is the TCP/IP port the server listens on, and is what you should use in the Sprite's URI. The *`device`* indicates what target device should be used. For USB devices use `utap` for COP/OnCE and `utap_dpi` for BDM or DPI. For Ethernet devices use `powertap` for COP/OnCE and `powertap_dpi` for BDM or DPI. If you have multiple devices of you can append a :*`serial-number`* to the USB *`device`* name. The eight-digit *`serial-number`* is located on the underside of the TAP device just after the revision information. For Ethernet devices append the device's IP address.

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

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

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

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

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

```
% config cc utap
```

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

```
% config cc utap_dpi
```

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

7.5.2. Common CCS Errors

Here are some common error messages and their causes:

Cable disconnected	The target board is not powered up, the board hardware is faulty or in a bad state or the jumper settings are incorrect.
CC not present	The required Command Converter is not present. You did not use `utap_bdm` or `utap_dpi` to connect CCS to a BDM or DPI TAP device connection.

Core not responding	CCS is no longer has control of the target system. The board hardware is faulty or in a bad state, the board initialization settings are incorrect or there is another debugger configuration problem.
USB open failure	For a Windows host, the USB driver on the host computer is hung. Unplug/replug the USB tap, or reboot the host PC if the problem persists. This might also happen if the USB drivers were not installed. You may install USB drivers manually from `m68k-elf\ccs\drivers` subdirectory of your Sourcery G++ Lite installation. For a Linux host this can occur if the permissions are not set correctly. Try running CCS as root, and if this resolves the problem, review the instructions in `m68k-elf/ccs/drivers/usb` subdirectory of your Sourcery G++ Lite installation for setting up USB permissions.
Maximum number of Command Converters reached	You have tried to reconfigure without first deleting the current configuration.
Cannot reset to debug mode	This can indicate that the clock speed is too high. Try a lower clock speed with the `speed=` option in the device URL.

7.6. Turbo BDM Light ColdFire Devices

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

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

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

There are no further options for the TBLCF device.

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

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

If this happens, you must install the driver for the TBLCF device, included with Sourcery G++ Lite. See Section 7.6.1, “Installing TBLCF (AxBDM) Windows Drivers” for installation instructions.

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

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

If you see this message, consult Section 7.6.2, “Configuring TBLCF (AxBDM) Devices on Linux” for configuration instructions.

7.6.1. Installing TBLCF (AxBDM) Windows Drivers

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

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

Windows may auto-detect the TBLCF device when it is connected, and invoke the driver installation dialog automatically. If you have already installed Sourcery G++ Lite, you may continue with the dialog using steps similar to those outlined above. Otherwise, close the dialog, install Sourcery G++ Lite first, and then follow the above steps to install the driver.

7.6.2. Configuring TBLCF (AxBDM) Devices on Linux

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

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

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

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

7.6.2.1. Configuring TBLCF with udev

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

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

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

7.6.2.2. Configuring TBLCF with hotplug

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

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

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

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

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

7.6.2.3. Troubleshooting TBLCF Device Permissions

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

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

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

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

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

7.7. 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:

```
> m68k-elf-sprite -l :10000 device-url board-file
```

starts the Sprite listening on port 10000. Use the following command to connect GDB to the remote Sprite:

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

to connect to the remote Sprite, 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.

7.8. Implementation Details

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

7.8.1. Software Breakpoints

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

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

```
handle SIGTRAP nostop noprint nopass
```

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

7.8.2. Hardware Watchpoints

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

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

7.8.3. Single Stepping

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

7.9. Supported Board Files

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

Board	**Config**	**Board**	**Config**
m51qe128evb Eval (20MHz)	`m51qe128evb-20`	m52223evb Eval (66MHz)	`m52223evb-66`
m51qe128evb Eval (40MHz)	`m51qe128evb-40`	m52223evb Eval (80MHz)	`m52223evb-80`
m51qe128evb Eval (50.33MHz)	`m51qe128evb-50`	m52235evb Eval (60MHz)	`m52235evb`
m51qe32evb Eval (20MHz)	`m51qe32evb-20`	m5235evb Eval	`m5235evb`
m51qe32evb Eval (40MHz)	`m51qe32evb-40`	m5249c3 Eval	`m5249c3`
m51qe32evb Eval (50.33MHz)	`m51qe32evb-50`	m5253evb Eval	`m5253evb`
m51qe64evb Eval (20MHz)	`m51qe64evb-20`	m5272c3 Eval	`m5272c3`
m51qe64evb Eval (40MHz)	`m51qe64evb-40`	m5275evb Eval	`m5275evb`
m51qe64evb Eval (50.33MHz)	`m51qe64evb-50`	m5282evb Eval	`m5282evb`
m5206ec3 Eval	`m5206ec3`	m5307c3 Eval	`m5307c3`
m5208evb Eval	`m5208evb`	m5329evb Eval	`m5329evb`
m5208evb RevE Eval	`m5208evbe`	m54455evb Eval	`m54455evb`
m5213evb Eval (66MHz)	`m5213evb-66`	m5485evb Eval	`m5485evb`
m5213evb Eval (80MHz)	`m5213evb-80`		

7.10. Board File Syntax

The *`board-file`* can be a user-written XML file to describe a non-standard board. The Sourcery G++ Debug Sprite searches for board files in the `m68k-elf/lib/boards` directory in the installation. Refer to the files in that directory for examples.

The file's DTD is:

```
<!-- Board description files

     Copyright 2007, 2008 CodeSourcery.  All rights reserved.
     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?)>
<!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>
```

```
<!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>`, `<features>`, `<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 the following `<property>` elements:	
	`cache`	This boolean property is used to indicate that the target has a cache. This knowledge is necessary to correctly write to a program's instruction stream.
	`floating-point`	This boolean property indicates whether floating point registers are provided on the target.
`<initialize>`	The `<initialize>` element allows board devices to be initialized before any attempt is made to download a program to it. It can contain `<write-register>`, `<write-memory>` and `<delay>` elements.	
`<feature>`	This element is used to inform GDB about additional registers and peripherals available on the board. It is passed directly to GDB; see the GDB manual for further details.	
`<memory-map>`	This element describes the memory map of the target board. It is used by GDB to determine where software breakpoints may be used and when flash programming sequences must be used. This element can occur at most once. It can contain `<memory-device>` elements.	
`<memory-device>`	This element specifies a region of memory. It has four attributes: `address`, `size`, `type` and `device`. The `address` and `size` attributes specify the location of the memory device. The `type` attribute specifies that device as `ram`, `rom` or `flash`. The `device` attribute is required for `flash` regions; it specifies the flash device type. The `<memory-device>` element can contain a `<description>` element. It can also contain the following `<property>` elements for additional flash-specific information:	
	`system-clock`	This numeric property is used for `cfm` flash devices. It specifies the target frequency, and used to determine the flash frequency divider value.
`<write-register>`	This element writes a value to a control register. It has three attributes: `address`, `value` and `bits`. The `bits` attribute is optional and defaults to 32. The address may be specified as a number, or as a name. The following registers are available: `CACR`, `ASID`, `ACR0`, `ACR1`, `ACR1`, `ACR1`,	

MMUBAR, VBR, ROMBAR0, ROMBAR1, FLASHBAR, RAMBAR0, RAMBAR1, MPCR, EDRAMBAR, SECMBAR, MBAR2, MBAR.

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

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

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

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

Chapter 8
Next Steps with Sourcery G++

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

8.1. 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*.

8.2. 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++-m68k-elf/` subdirectory of your installation directory.

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

```
> cd $INSTALL/share/doc/sourceryg++-m68k-elf/man/man1
```

Then you can invoke **man** as:

```
> man ./m68k-elf-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/