

SoftConsole v3.4 Release Notes

Revision: 1.0

Date: March 25th 2013

TABLE OF CONTENTS

1	INTRODUCTION	4
2	WHAT'S NEW	4
2.1	COMPONENT TOOL VERSIONS.....	4
2.2	NEW FEATURES.....	5
2.2.1	<i>Support for Microsemi SmartFusion2 cSoC</i>	5
2.2.2	<i>Consolidated debug support for Cortex-M3 targets</i>	5
2.2.3	<i>Fixes and improvements</i>	5
2.3	FIXES AND IMPROVEMENTS	5
3	USING SOFTCONSOLE V3.4 WITH SMARTFUSION2	5
3.1	SMARTFUSION2 BOARD JTAG_SEL CONFIGURATION	5
3.2	TARGETING SMARTFUSION2 CORTEX-M3.....	5
3.3	CREATING PROGRAMS	9
3.4	DRIVERS_CONFIG/SYS_CONFIG.....	13
3.5	CONSOLIDATED DEBUG SUPPORT FOR CORTEX-M3 TARGETS	13
4	CORTEX-M3 LINKER SCRIPTS	15
4.1	OVERVIEW	15
4.2	LINKER SCRIPTS	15
4.2.1	<i>Linker scripts bundled with the CMSIS</i>	15
4.2.2	<i>Other linker scripts</i>	16
4.2.3	<i>User modified linker scripts</i>	16
4.2.3.1	debug-in-actel-smartfusion-envm.ld debug-in-microsemi-smartfusion2-envm.ld.....	16
4.2.3.2	debug-in-actel-smartfusion-esram.ld debug-in-microsemi-smartfusion2-esram.ld	17
4.2.3.3	debug-in-external-ram.ld.....	17
4.2.3.4	production-execute-in-place.ld.....	17
4.2.3.5	production-relocate-executable.ld.....	17
4.2.3.6	debug-in-external-flash.ld	18
4.2.4	<i>Configuring the linker script for your project</i>	18
5	KNOWN LIMITATIONS, ISSUES, AND WORKAROUNDS	19
5.1	GENERAL	19
5.2	FLASH PROGRAMMING/PROGRAM DOWNLOAD	23
5.3	CORTEX-M3 DEBUGGING.....	24
5.4	CORE8051S.....	25
6	DOCUMENTATION	29
7	SYSTEM REQUIREMENTS	29
7.1	SUPPORTED PLATFORMS	29
7.2	SOFTWARE ENVIRONMENT	30
8	LICENSING	30

1 Introduction

SoftConsole v3.4 adds support for Microsemi SmartFusion2 cSoC so that the list processor targets now supported by SoftConsole is:

- Microsemi SmartFusion2[®] MSS ARM[®] Cortex-M3
- Microsemi SmartFusion[®] MSS ARM[®] Cortex-M3
- Microsemi ARM[®] Cortex-M1
- Microsemi CoreMP7 (ARM7TDMI-S™)
- Microsemi Core8051s

2 What's New

2.1 Component Tool Versions

SoftConsole v3.4 comprises the component tool packages listed below.

- Eclipse Platform (and Java Runtime)
 - Eclipse IDE Galileo SR2 v3.5.2
 - Eclipse CDT (C/C++ Development Tooling) v6.0.2
 - Oracle Java 6 Standard Edition Version 6 Update 22 (1.6.0_22-b04)
- CodeSourcery Sourcery G++ Lite for ARM EABI (GNU Toolchain for ARM Processors) 2010q1-188
 - GCC (The GNU Compiler Collection) C and C++ Compilers v4.4.1-sg++
 - GDB (The GNU Project Debugger) v7.0.50-sg++ (plus Microsemi modifications v1.2)
 - GNU Binutils (Binary Utilities) v2.19.51-sg++
 - GNU Make v3.81
 - Newlib C standard library v1.17.0-sg++
- 8051 Development Tools
 - SDCC Small Device C Compiler v2.6.3 (source code available on request from Microsemi)
 - CodeSourcery omf2elf (OMF to ELF converter) v1.0-7
 - GDB (The GNU Project Debugger) v6.7.50-sg++ (plus Microsemi modifications v1.1)
 - GNU Binutils (Binary Utilities) v2.18.50-sg++ (plus Microsemi modifications v1.1)
 - GDB and Binutils are based on the CodeSourcery G++ Lite 2008q1-126 versions
- Hardware target debug support tools – the following tools sit between GDB and the relevant target processor in order to facilitate hardware-based debugging by translating between GDB Remote Serial Protocol and target-processor-specific JTAG/debug commands/responses:
 - Cortex-M1/Cortex-M3: CodeSourcery ARM Debug Sprite v1.0-8 + Microsemi v1.3.8-M3 + Flash Programming v1.2.2

- Core8051s: C8051 Debug Sprite v1.0-8 + Microsemi v1.6.0 + Flash Programming v1.2.2
- CoreMP7: FS2 In-Target System Analyzer for ARM Processor Cores v1.2.0

2.2 New Features

2.2.1 Support for Microsemi SmartFusion2 cSoC

Support for development and debugging of software for the SmartFusion2 MSS Cortex-M3 in embedded SRAM (ESRAM) and embedded flash (ENVM).

2.2.2 Consolidated debug support for Cortex-M3 targets

SoftConsole now provides a single debug launch configuration for both SmartFusion Cortex-M3 (r1p1) and SmartFusion2 Cortex-M3 (r2p1). This single debug launch configuration also supports all sample link scenarios that are supported by the linker scripts bundled with the SmartFusion CMSIS-PAL and SmartFusion2 CMSIS and Hardware Abstraction Layer firmware cores. This simplifies debugging by obviating the need to match a specific debug launch configuration to the linker script used to build the program. There is also a single debug sprite which seamlessly supports both SmartFusion Cortex-M3 and SmartFusion2 Cortex-M3.

2.2.3 Fixes and improvements

Various SAR (Software Action Request) feature, usability and bug issues have been fixed as detailed below.

2.3 Fixes and Improvements

The following Software Action Request (SAR) issues are fixed.

SAR Number	Description
18210	Add support for further Actel board flash parts & Fusion2 ENVM
29618	Installer: remove or refine checks for FlashPro software/drivers
30301	8051: assembler build step missing from project settings
41646	unable to debug/program the SmartFusion2 device

3 Using SoftConsole v3.4 with SmartFusion2

3.1 SmartFusion2 board JTAG_SEL configuration

Ensure that the SmartFusion2 board JTAG_SEL switch/jumper is set to H in order to allow the SoftConsole debug sprite connect to the target.

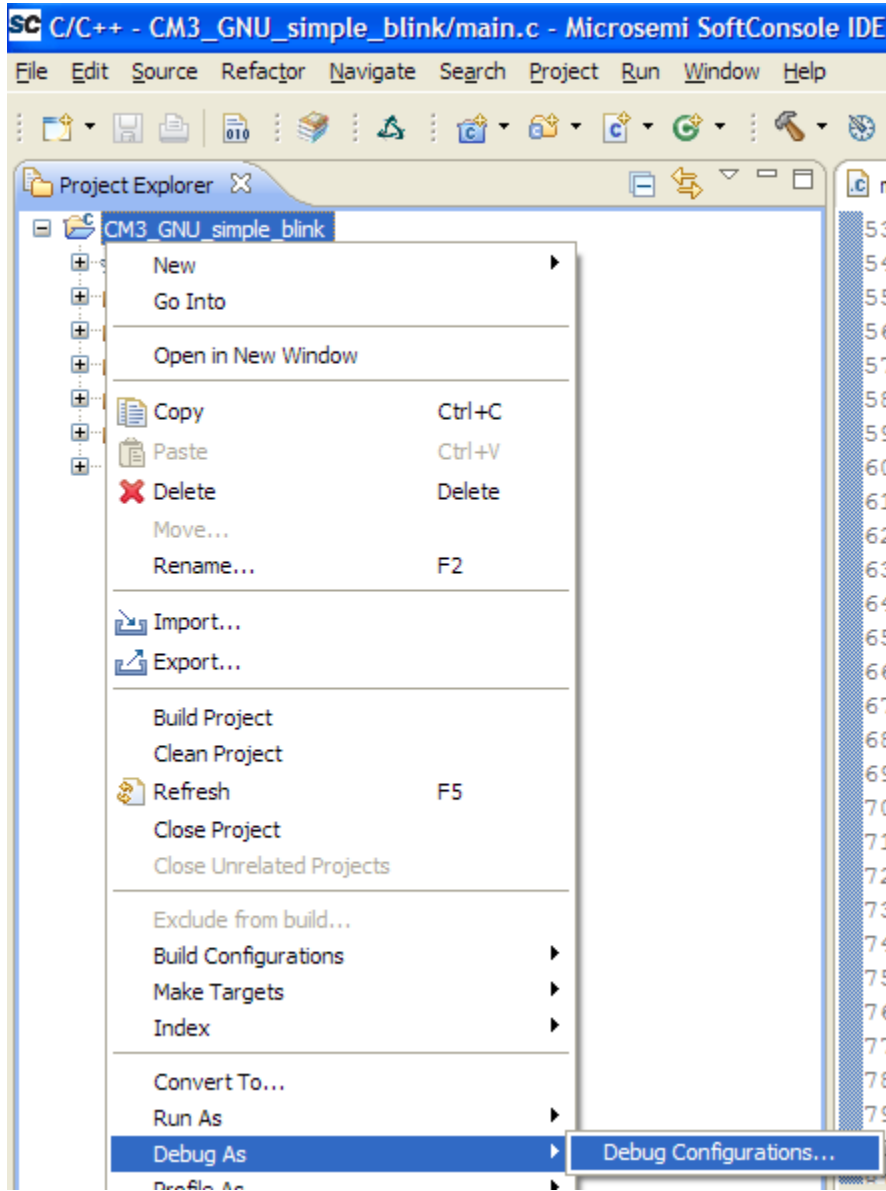
3.2 Targeting SmartFusion2 Cortex-M3

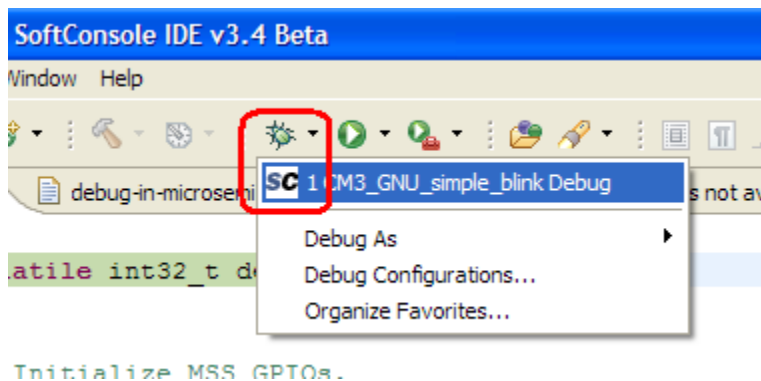
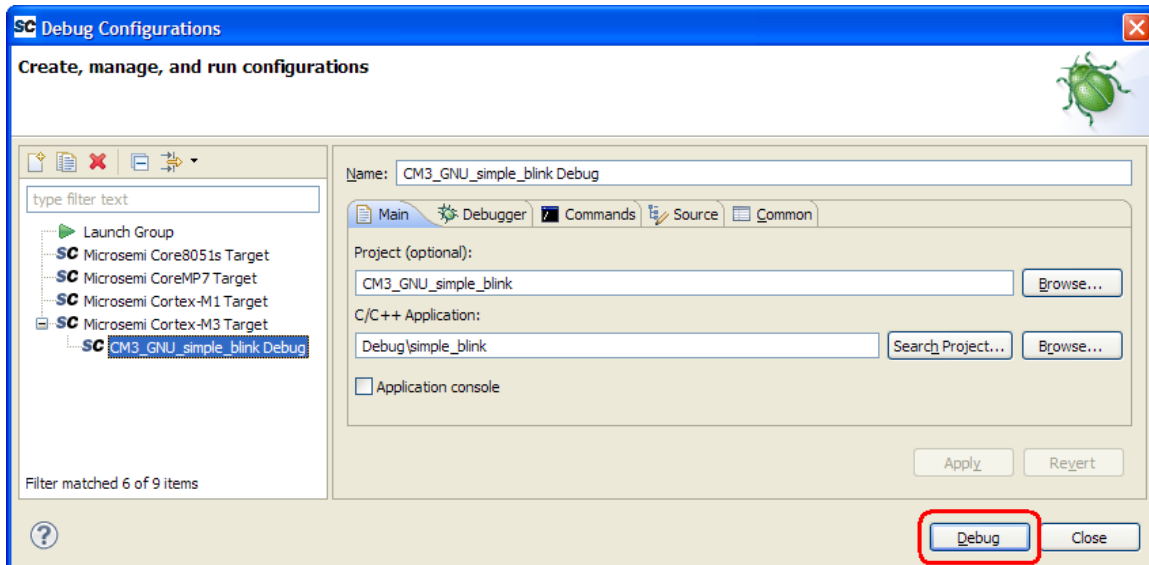
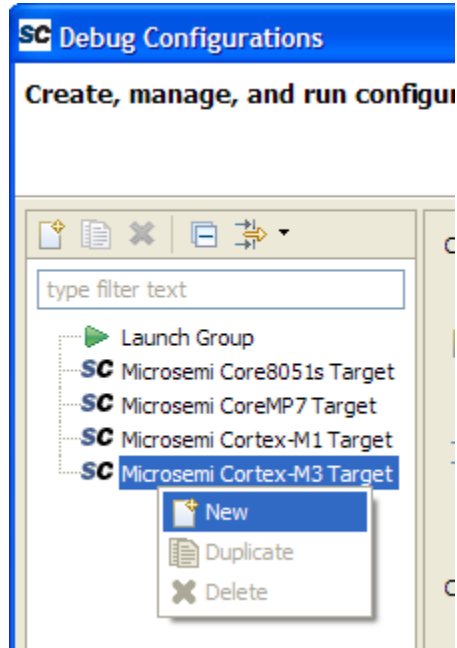
To target SmartFusion2 Cortex-M3 compile your program as normal and link using the appropriate SmartFusion2 MSS CMSIS and Hardware Abstraction Layer sample linker script. For example:

SmartFusion2 MSS CMSIS and Hardware Abstraction Layer sample linker script	Description
debug-in-microsemi-smartfusion2-esram.ld	Download to and debug from SmartFusion2 embedded SRAM (ESRAM) at 0x20000000.

debug-in-microsemi-smartfusion2-envm.ld	Download to and debug from SmartFusion2 embedded flash/NVM (ENVM) at 0x60000000 mirrored to 0x00000000 at runtime. Uses the SmartFusion2 ENVM flash programming profile (C:\Program Files\Microsemi\SoftConsole v3.4\Sourcery-G++\share\sprite\flash\microsemi-smartfusion2-envm.xml) to download the program to ENVM. Programs downloaded in this way will also persist in ENVM so that they run from ENVM from power on reset.
production-execute-in-place.ld production-relocate-executable.ld	These linker scripts are for linking “production” programs which will be loaded to ENVM using a FlashPro ENVM data storage client and which will execute directly from ENVM or relocate/copy to and run from ESRAM. They are not designed for interactive debugging using SoftConsole.
debug-in-external-ram.ld debug-in-external-flash.ld	Please note that download to/debugging from external memories at/above 0x70000000 has not yet been exercised and tested on SmartFusion2.

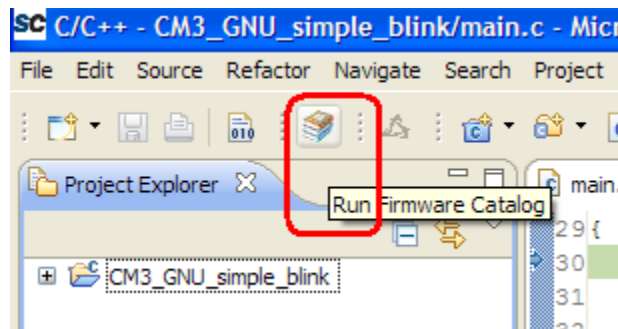
To debug a program compiled for download/debug to ESRAM or ENVM simply create a *Microsemi Cortex-M3 Target* debug launch configuration and run it. SoftConsole will connect to the target, download the program and stop at a temporary breakpoint at `main()` at which point you can interactively debug your program. Note that even if you change the linker script used to link your program you can continue to use the same debug launch configuration unlike in SoftConsole v3.3 where you had to match the debug launch configuration to the linker script.



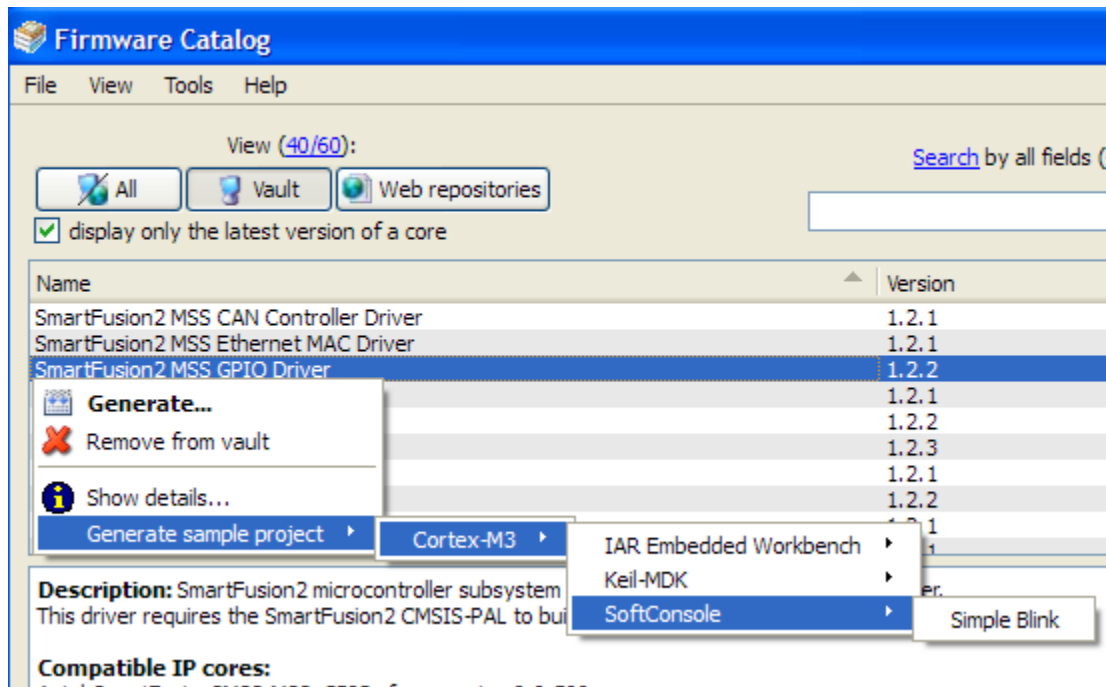


3.3 Creating programs

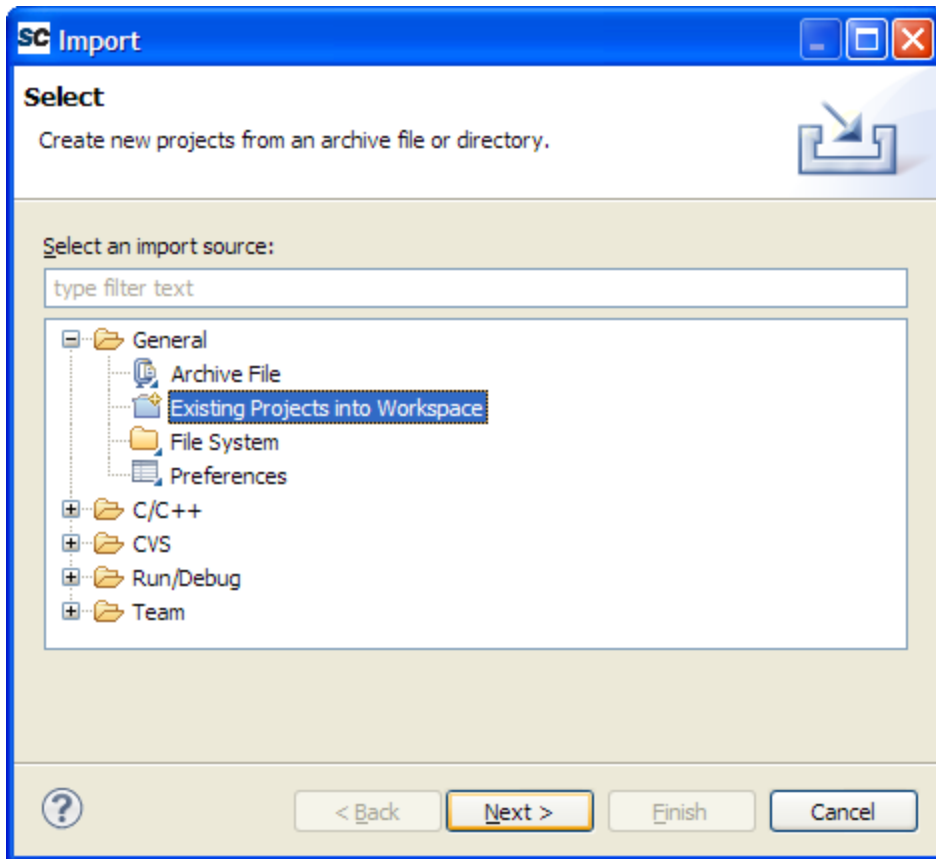
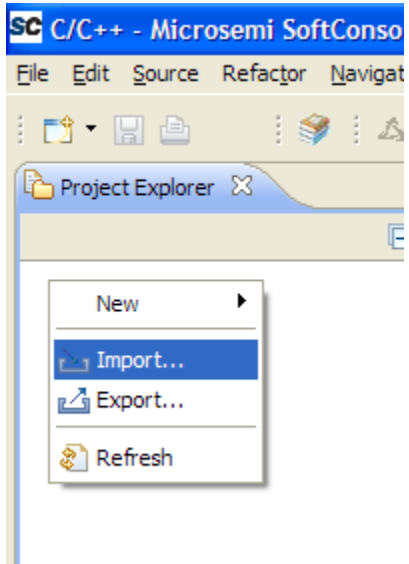
The easiest way to get started with SmartFusion2 Cortex-M3 software development and debugging is to launch the Firmware Catalog from SoftConsole:

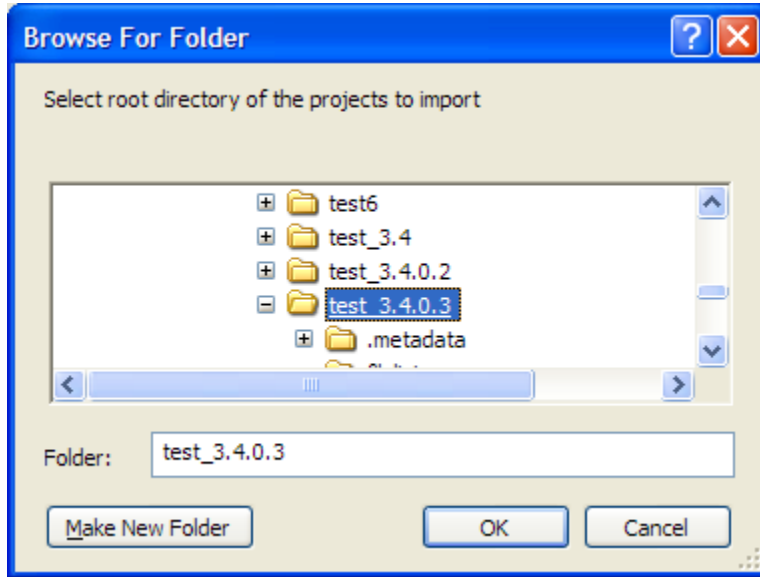


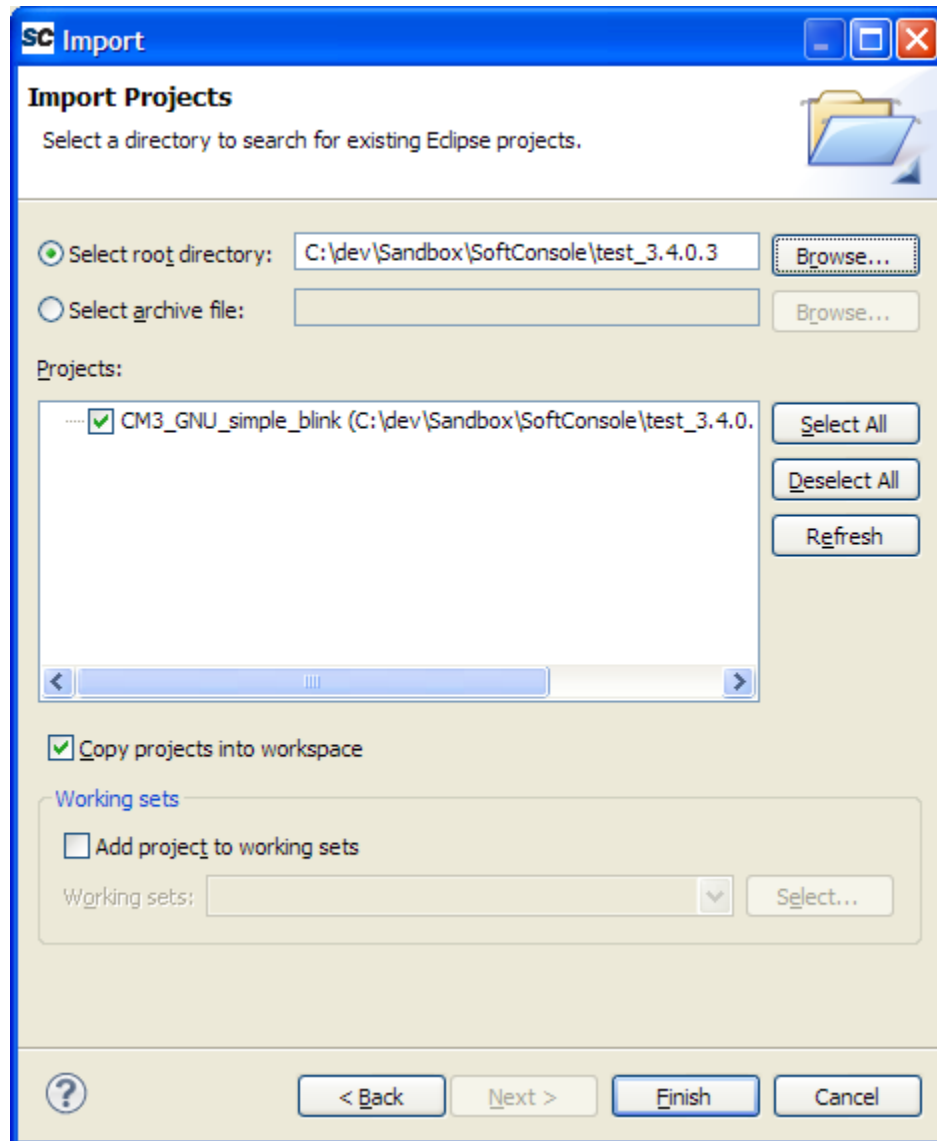
and generate one of the sample projects from the Firmware Catalog – for example:



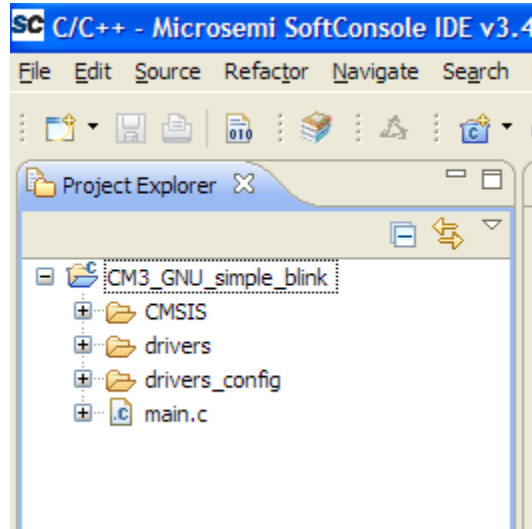
And then import it into the workspace:







And then the project will appear in your workspace:



You can also build programs up from scratch by generating the required firmware cores (for example SmartFusion2 CMSIS and Hardware Abstraction Layer and relevant drivers) into a new project and writing your own code.

Alternatively you can use the SoftConsole workspace and projects created by the integrated flow in Libero SoC.

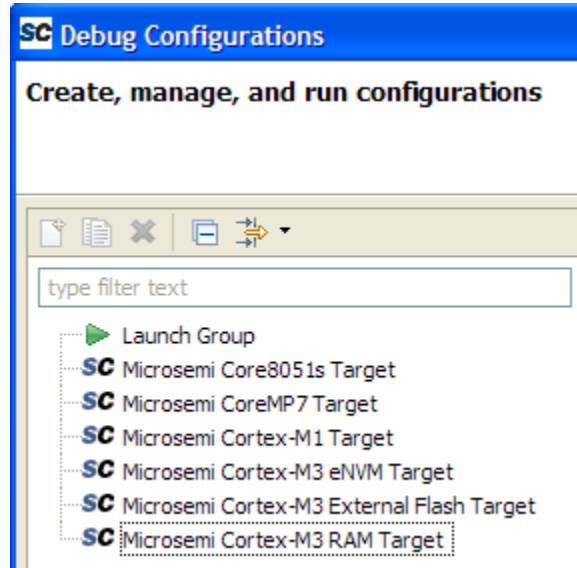
3.4 drivers_config/sys_config

In order to avoid a software/target hardware mismatch you must ensure that your SoftConsole project contains the `drivers_config/sys_config` folder/files created by Libero SoC when generating the MSS for your design.

3.5 Consolidated debug support for Cortex-M3 targets

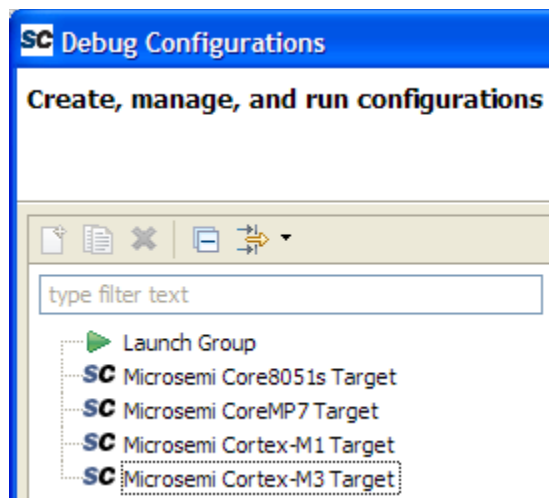
Previous versions of SoftConsole had several different debug launch configurations for Cortex-M3 which had to be matched to the sample CMSIS linker script used to build the program – for example:

Debug launch configuration	SmartFusion MSS CMSIS-PAL sample linker script
Microsemi Cortex-M3 ENVM Target	debug-in-actel-smartfusion-envm.ld
Microsemi Cortex-M3 External Flash Target	debug-in-external-flash.ld
Microsemi Cortex-M3 RAM Target	debug-in-actel-smartfusion-esram.ld debug-in-external-ram.ld



Mismatching the debug launch configuration and linker script would lead to debug problems.

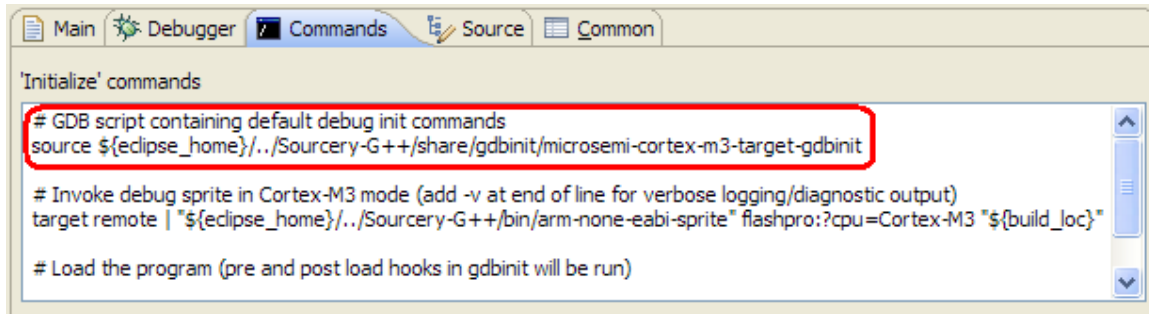
SoftConsole v3.4 provides consolidated debug support for all Cortex-M3 targets and link scenarios. This means that there is a single debug sprite and a single debug launch configuration for Cortex-M3 regardless of the target device (SmartFusion Cortex-M3 r1p1 or SmartFusion2 Cortex-M3 r2p1) or linker script used. This significantly simplifies the SoftConsole Cortex-M3 debugging experience.



When the debug sprite connects to the target hardware it probes the FPGA and Cortex-M3 to figure out whether it needs to send the SmartFusion CTXSELECT (0x94) or the SmartFusion2 M3DEBUG (0x0A) JTAG command in order to bypass the FPGA JTAG controller and talk directly to the Cortex-M3 for debugging purposes. It also probes the Cortex-M3 CPUID base register to understand if it is communicating with the SmartFusion Cortex-M3 r1p1 or the SmartFusion2 Cortex-M3 r2p1.

The single debug launch configuration makes use of a GDB script (C:\Program Files\Microsemi\SoftConsole v3.4\Sourcery-G++\share\gdbinit\microsemi-cortex-m3-target-gdbinit) to manage debug related differences between the SmartFusion and SmartFusion Cortex-M3 targets. This GDB script contains pre and post GDB load

command hooks that cater for the two targets and the common sample link/debug scenarios supported by the CMSIS firmware cores for each target. The GDB script is referenced from the debug launch configuration debug initialization commands.



```
'Initialize' commands
# GDB script containing default debug init commands
source ${eclipse_home}/../Sourcery-G++/share/gdbinit/microsemi-cortex-m3-target-gdbinit
# Invoke debug sprite in Cortex-M3 mode (add -v at end of line for verbose logging/diagnostic output)
target remote | "${eclipse_home}/../Sourcery-G++/bin/arm-none-eabi-sprite" flashpro:?cpu=Cortex-M3 "${build_loc}"
# Load the program (pre and post load hooks in gdbinit will be run)
```

In cases where a customized linker script is used it may be necessary to use a (copied and) modified version of this script.

4 Cortex-M3 Linker Scripts

4.1 Overview

This section summarises the example linker scripts provided by the SmartFusion CMSIS-PAL and SmartFusion2 CMSIS Hardware Abstraction Layer firmware cores. Refer to the CMSIS documentation and example projects for more information about these and other CMSIS features and capabilities.

4.2 Linker scripts

4.2.1 Linker scripts bundled with the CMSIS

The SmartFusion2 CMSIS Hardware Abstraction Layer firmware core bundles the following linker scripts, which can be used to build/link SmartFusion2 Cortex-M3 executables and are described in more detail below:

- debug-in-microsemi-smartfusion2-envm.ld
- debug-in-microsemi-smartfusion2-esram.ld
- debug-in-external-ram.ld
- production-execute-in-place.ld
- production-relocate-executable.ld

Similarly the SmartFusion CMSIS-PAL firmware core bundles the following linker scripts, which can be used to build/link SmartFusion Cortex-M3 executables and are described in more detail below:

- debug-in-actel-smartfusion-envm.ld
- debug-in-actel-smartfusion-esram.ld
- debug-in-external-ram.ld
- production-execute-in-place.ld
- production-relocate-executable.ld

The `debug-in-*` linker scripts are for building/linking programs for downloading to and debugging from ENVN, ESRAM and external RAM.

The `production-*` linker scripts are for building/linking “production” executable images which will execute out of reset in a live system and which are stored in ENVN not by SoftConsole but through some other mechanism – e.g. SmartFusion/SmartFusion2 ENVN Data Storage Client using the Intel Hex (`<project-name>.hex`) file generated by SoftConsole for the program image in the project’s Debug or Release folder.

Programs built using the `debug-in-actel-smartfusion-envn.ld/debug-in-microsemi-smartfusion2-envn.ld` linker scripts can be downloaded to and debugged in ENVN. Once they have been downloaded to ENVN they can also run out of ENVN from power on reset.

4.2.2 Other linker scripts

The following sample linker script is bundled with the SoftConsole installation:

- `debug-in-external-flash.ld`

This is installed as `<SoftConsole-install-folder>\src\Cortex-M3\linker-script-examples\debug-in-external-flash.ld`. For example, on most systems this will be `C:\Program Files\Microsemi\SoftConsole v3.4\src\Cortex-M3\linker-script-examples\debug-in-external-flash.ld`.

4.2.3 User modified linker scripts

The combination of linker scripts bundled with the CMSIS and the debug launch configurations supported by SoftConsole cover a wide range of common development and debug situations. However some targets may require the adaptation of these to suit the specific platform being targeted. If you need to modify a linker script to a target your specific hardware platform then please refer to the GNU ld linker (http://en.wikipedia.org/wiki/GNU_linker) documentation for information.

4.2.3.1 debug-in-actel-smartfusion-envn.ld debug-in-microsemi-smartfusion2-envn.ld

Use these linker scripts in order to build/link programs for downloading to and debugging from ENVN.

They cater for 256KB of ENVN @ 0x60000000, which is also mirrored @ 0x00000000, and 64KB of ESRAM @ 0x20000000.¹ The program code is downloaded to ENVN @ 0x60000000 and executed from ENVN mirrored @ 0x00000000.² ESRAM @ 0x20000000 is used for data (BSS/zero initialized data, static initialized data, stack and heap).

¹ Note that the sample linker scripts described here may need modification if the target device supports a different amount of ENVN or ESRAM.

² In this linker script the GNU ld linker LMA (Load Memory Address) is 0x60000000 while the VMA (Virtual Memory Address), at which the code actually executes, is 0x00000000. The reason for this is twofold – (a) the program cannot be downloaded directly to ENVN mirrored @ 0x00000000 because ENVN is not writeable (even through the debug interface) at that address and (b) Cortex-M3 only supports hardware breakpoints below 0x20000000 so the code must execute below this address in order to support debugging. Note also that the linker script defines `__mirrored_nvn = 1` in order to inform the CMSIS not to do any copying of code from LMA to VMA addresses since this is already taken care of by the mirroring of ENVN from 0x60000000 to 0x00000000.

4.2.3.2 debug-in-actel-smartfusion-esram.ld debug-in-microsemi-smartfusion2-esram.ld

Use these linker scripts in order to build/link programs for downloading to and debugging from ESRAM. It caters for 64KB ESRAM @ 0x20000000 in which both the program code and data (BSS/zero initialized data, static initialized data, stack and heap) reside.

4.2.3.3 debug-in-external-ram.ld

Use this linker script in order to build/link programs for downloading to and debugging from external RAM @ the SmartFusion Cortex-M3 External Memory Type 0 address of 0x70000000.

It caters for 2MB of external RAM @ 0x70000000³ and 64KB of ESRAM @ 0x20000000. The program code is downloaded to external RAM @ 0x70000000 with the exception of the vector table which is downloaded to ESRAM @ 0x20000000.⁴ Data (BSS/zero initialized data, static initialized data, stack and heap) also reside in ESRAM @ 0x20000000.

4.2.3.4 production-execute-in-place.ld

Use this linker script in order to build/link programs for execution from ENVM @ 0x00000000 out of reset. This is for production programs which are stored in and execute directly from ENVM.⁵

It caters for 1MB of ENVM @ 0x00000000 in which the program code is stored and 64KB of ESRAM @ 0x20000000 in which data (BSS/zero initialized data, static initialized data, stack and heap) resides.

Programs linked using this linker script cannot be downloaded or debugged using SoftConsole and the program image must be stored in ENVM using some other mechanism such as an ENVM Data Storage Client. In this case the ENVM data storage client can be configured to read in hex file version of the program image produced by the SoftConsole build process. The ENVM data storage client can then be programmed with the program image using Libero/FlashPro.

4.2.3.5 production-relocate-executable.ld

Use this linker script in order to build/link programs that will execute from reset out of ENVM but which will be copied/relocated to external RAM @ 0x70000000 by the CMSIS startup code.

It caters for 1MB of ENVM @ 0x00000000, 2MB of external RAM @ 0x70000000 and 64KB of ESRAM @ 0x20000000. The program code resides in ENVM @ 0x00000000 and the CMSIS startup code copies/relocates it to external RAM @ 0x70000000 (excluding the vector table which remains resident in ENVM @ 0x00000000). Data (BSS/zero initialized data, static initialized data, stack and heap) resides in ESRAM @ 0x20000000.

³ If SmartFusion external RAM is not at 0x70000000 (e.g. @ External Memory Type 1 address 0x74000000 or not at the “bottom” of External Memory Type 0 @ 0x70000000 or External Memory Type 1 @ 0x74000000) then the linker script and *Actel Cortex-M3 RAM Target* debug initialization commands will require modification.

⁴ Cortex-M3 requires that the vector table resides below 0x40000000 in the Code (0x00000000-0x1FFFFFFF) or SRAM (0x20000000-0x3FFFFFFF) memory regions.

⁵ Building a program with the `debug-in-actel-smartfusion-envm.ld/ debug-in-microsemi-smartfusion2-envm.ld` linker script and downloading it to ENVM also means that the program will thereafter run from power on reset. However the rationale for the `production-execute-in-place.ld` linker script is that in certain production scenarios it is not feasible to use SoftConsole to download the program but it is preferable to be able to fully program the target using a single STAPL file containing all relevant information (e.g. SmartFusion/SmartFusion2 MSS configuration data, other ENVM content, Cortex-M3 program image for ENVM, FPGA fabric design etc.).

As with the `production-execute-in-place.ld` linker script, programs linked using this linker script cannot be downloaded or debugged using SoftConsole and the program image must be stored in ENVM using some other mechanism such as an ENVM data storage client.

4.2.3.6 debug-in-external-flash.ld

A further linker script, `debug-in-external-flash.ld`, is installed in the SoftConsole tree under `src\Cortex-M3\linker-script-examples`. A version of this script will be part of the CMSIS firmware core in the future.

Use this linker script in order to build/link programs for downloading to external flash. It is not possible to actually debug the code which is executing in external flash, since its starting location of `0x74000000` is beyond the Cortex-M3 requirement that hardware breakpoints be placed beneath `0x20000000`.

It caters for 16MB external flash across two devices: the first 8MB @ `0x74000000` and the second 8MB @ `0x74800000`. Code in the `.text` section is placed in the external flash. The `.init` section containing the vector table is downloaded to ENVM @ `0x60000000` and executed from ENVM mirrored @ `0x00000000`.⁶ ESRAM @ `0x20000000` is used for data (BSS/zero initialized data, static initialized data, stack and heap), with its initial contents copied from the external flash.

The project must be compiled with the `-mlong-calls` flag in order for the startup code to be able to jump up to `main()` which will be located in the external flash. Select the project in *Project Explorer*, choose *Properties > C/C++ Build > Settings > GCC C Compiler*, and add `-mlong-calls` into the *Command* field. Alternatively this option can be applied to just the `./CMSIS/startup_gcc/startup_a2fxxxm3.s` file rather than the project as a whole.

Note: The second external flash device at `0x74800000` is expected to be already unlocked when SoftConsole performs the programming of the flash memory.

Note: Currently this linker script cannot support C++ programs because the `.ARM.exidx` section is beyond the 31-bit range for the linker to provide the required relocations from the `.text` section in the external flash.

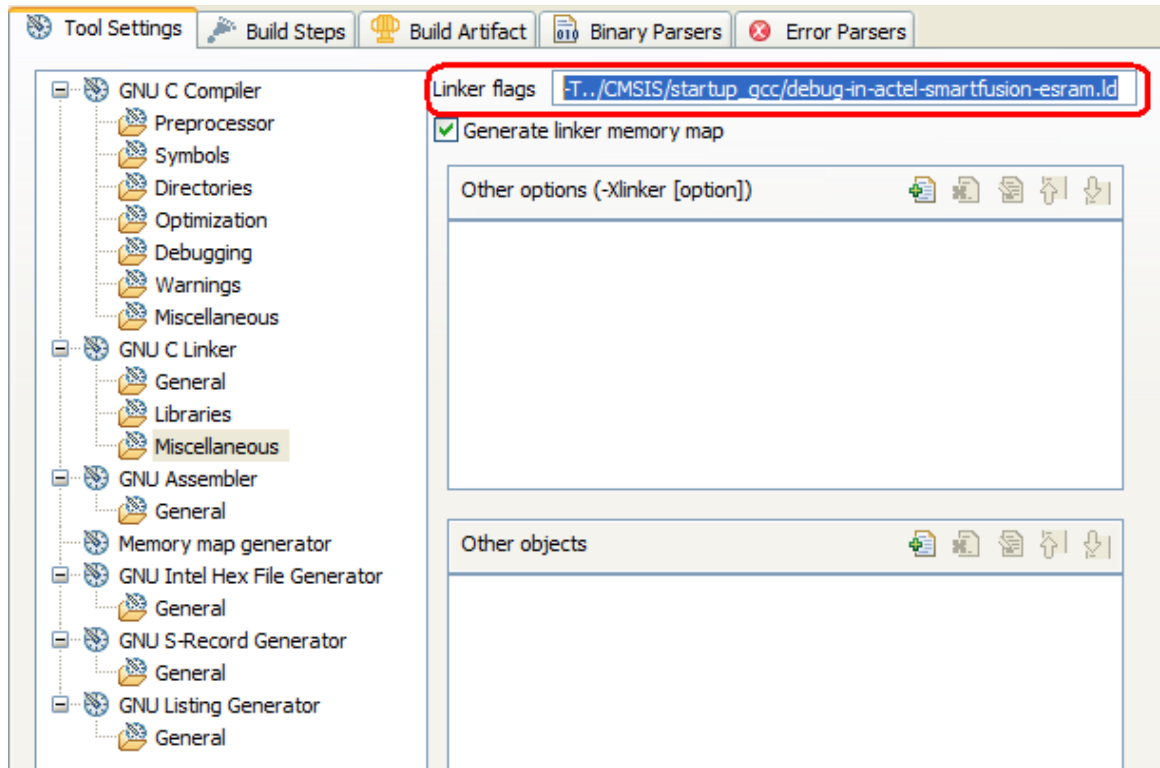
4.2.4 Configuring the linker script for your project

To configure or change the linker script used when building a project:

- Right click on the project in the *Project Explorer* view and choose *Properties*⁷
- In the **Properties for <project-name>** dialog navigate to **C/C++ Build > Settings > GNU C Linker > Miscellaneous**
- Ensure that the **Linker flags** field has the appropriate linker script specified using the GNU ld linker's `-T <ld-script>` option – e.g.:

⁶ As in other linker scripts, the GNU ld linker LMA (Load Memory Address) is `0x60000000` while the VMA (Virtual Memory Address), at which the code actually executes, is `0x00000000`. The reason for this is twofold – (a) the program cannot be downloaded directly to eNVM mirrored @ `0x00000000` because ENVM is not writeable (even through the debug interface) at that address and (b) Cortex-M3 only supports hardware breakpoints below `0x20000000` so the code must execute below this address in order to support debugging. Note also that the linker script defines `__mirrored_nvm = 1` in order to inform the CMSIS not to do any copying of code from LMA to VMA addresses since this is already taken care of by the mirroring or eNVM from `0x60000000` to `0x00000000`.

⁷ Project properties must be modified separately for Debug and Release build configurations.



5 Known Limitations, Issues, and Workarounds

5.1 General

Build problems and bare.specs: If you have problems compiling/building your project when using the SmartFusion CMSIS-PAL and peripheral driver firmware cores then remove any references to `bare.specs` in the project properties in case these are causing problems:

- Right-click the project in the **Project Explorer** and choose **Properties**
- In the **Properties for <project-name>** dialog navigate to **C/C++ Build**
- Under each of **GNU C Compiler**, **GNU C Linker** and **GNU Assembler** look at the **Command** field and if it contains `-specs=bare.specs` then remove this option as this is not relevant when building “bare metal” (non [RT]-OS) based programs using the SmartFusion CMSIS-PAL and peripheral driver firmware cores.

Cortex-M1 sample linker scripts no longer bundled with SoftConsole: Please note that sample linker scripts for the Cortex-M1 are no longer installed by default as part of SoftConsole. Instead they can now be obtained from the Cortex-M1 HAL using the Firmware Catalog.

10452 – Eclipse plugins and SoftConsole: The download and integration of arbitrary Eclipse plugins into SoftConsole (e.g. via **Help > Check for Updates and Help > Install New Software**) is not supported.

13017 – Memory Monitor memory writes happen at wrong granularity: Writing a value to memory using a *Memory Monitor* view takes place with the hardware only one byte at a time. Thus, setting a particular address to have a particular 32-bit value involves asking GDB to write four individual bytes. For some target designs, addresses are used assuming a single write will take place of all four bytes. In particular, APB memory may depend upon this behavior, where

you only need the lower 8 bits of the 32-bit word involved. Until this is fixed in a future release, make sure only a single one-byte write takes place for any given address: right-click in the memory region you want to modify and select **Format**. In the **Format** dialog, change the column size from 4 to 1, and click **OK**. You must make this change for each distinct memory monitor or window in your project if you need to change the width of your reads and writes.

13063 – Remove all terminated launches remains disabled even when relevant: When a GDB debugging session is terminated, the option to **Remove all terminated launches** may remain greyed out on both the toolbar and the right-click context menu even when some terminated launches remain listed.

13389 – Debug configuration Program Selection does not work if build target file has extension: When configuring a debug launch configuration for your program (right-click the project in the **Project Explorer**, choose **Debug As > Debug Configurations**, right-click the relevant target CPU and memory debug configuration, and choose **New**), the **Program Selection** dialog launched by the **Search Project** button will be blank if the executable built by the project has a file extension. The **Project Selection** dialog will only find executables with no file extension.

13956 – Prevent selecting a project directory for use as a workspace: Eclipse/CDT does not protect against the creation of a workspace inside a workspace or project folder and the recursive creation of very deeply nested folder structures that may result when importing files in such a context. To avoid this problem, never create a workspace inside another workspace or project folder, and never create a project inside another project folder.

17678 – Breakpoint from a different SoftConsole project is used in current project: If the same file is part of two open projects and a breakpoint is set in the file in one project, then when debugging the second project the processor stops at the breakpoint from the first project. A workaround is to close all projects other than the project you are debugging. To do this:

- Right-click the project in **Project Explorer** and select **Close Unrelated Projects**.

18968 – Step empty loop at C level resumes execution until loop completes & cannot be paused: GDB considers an empty loop to be the equivalent of a single statement or atomic instruction. For example, when the debugger is about to execute the code

```
volatile unsigned int i;
for (i = 0; i < ~0; i++) /* ← */
;
```

if you tell it to step on the loop, execution will resume until all iterations of the for loop have completed. Control only returns to the debugger once that has happened. However, for long delay loops like the above, this can take a very long time making it appear like the debug session has locked up. While the loop is executing, it is not possible to break into execution of the loop itself. The two options available are to: a) wait for execution to complete and control is returned to the debugger, or b) press the red Stop button to terminate the debug session.

21406 – Debugging does not work well for interrupt driven programs: While debugging a program on a target, GDB will not react if an interrupt occurs on the device and causes execution to go to an interrupt handler. Instead, SoftConsole will believe the program is still performing the steps the user requested, and thus not offer the Pause button to let you suspend execution. In this situation the only option is to terminate execution of the program.

22011 – Default character encoding can cause issues for non-Latin-1 character sets: The use of character sets for languages other than those considered “Western European” (of The Americas, Western Europe, Oceania, and much of Africa) is not yet supported. To change the

encoding for your files to the Unicode UTF-8 encoding, select **File > Properties** and change **Text file encoding to UTF-8**.

27949 – variadic functions compile OK but editor mistakenly reports syntax errors: due to a problem with CDT errors can be reported in the editor on variadic functions (using `va_arg()` etc.) even where they have compiled correctly. See here for more information: <http://www.eclipse.org/forums/index.php?t=msg&goto=223579>.

28697 – Microsemi Default code style formatter removes leading whitespace in comments: the Microsemi Default code formatter active by default (see **Window > Preferences > C/C++ > Code Style**) removes leading space in comments. For example using **Edit > Format** with the Microsemi Default code style formatter active changes this:

```
/* This line
   and this one. */
```

to this

```
/* This line
and this one. */
```

while it does not alter the whitespace in the following:

```
/* This line
 *   and this one
 *       and another one */
```

29172 – "Unresolved inclusion" annotation on system includes: If you import a project created with SoftConsole v3.1 or earlier into SoftConsole v3.2 or later then "system" header include directives (e.g. `#include <stdio.h>`) will be annotated with an "Unresolved inclusion" warning in the annotation margin on the left hand side of the editor window. In contrast a project created afresh with SoftConsole v3.2 or later will not exhibit this issue.

The reason for this is that between SoftConsole v3.1 and v3.2 (or, to be more accurate, between CDT v4.0.3 and v6.0.2) the way in which C/C++ project settings for system include files were handled was changed. In particular projects created with SoftConsole v3.2 automatically have project level settings specifying where "system" include files are to be found.

This warning is innocuous, can be ignored and does not impact the build process. However if you want to eliminate it then there are two options:

1. Recreate the project using SoftConsole v3.2 - create an empty project, import the folders/files from the original and then apply the project settings in the original to the new project. Because the project created with SoftConsole v3.2 or later has the relevant system include paths specified in the project settings automatically the "unresolved inclusion" annotations no longer appear.
2. Manually modify the system include path settings for the existing project as follows:
 - a. Right-click the project in **Project Explorer** and choose **Properties**
 - b. Go to **C/C++ General > Paths and Symbols > Includes**

- c. In the **Languages** list/tree view the following will be listed: **GNU C** and **s,S**. For both of these add the following Include directories (adjusting the specific paths if you have installed SoftConsole v3.4 in a non default location):
 - i. c:/program files/microsemi/softconsole v3.4/sourcery-g++/lib/gcc/arm-none-eabi/4.4.1/include
 - ii. c:/program files/microsemi/softconsole v3.4/sourcery-g++/lib/gcc/arm-none-eabi/4.4.1/include-fixed
 - iii. c:/program files/microsemi/softconsole v3.4/sourcery-g++/arm-none-eabi/include
- d. Click **OK** on the **Properties** dialog and when prompted to rebuild the project index click **Yes**. Any previously displayed system include "unresolved inclusion" annotations should now disappear.

29363 – Only a single debug session per host PC is supported: At the moment SoftConsole uses a single "global" named mutex to ensure that, at most, a single debug session is active at any one time. This is more restrictive than absolutely necessary and could be relaxed to allow for a maximum of one debug session per attached FlashPro programmer. To do this the mutex name should incorporate the FlashPro URI style name. This would allow for, say, two separate instances of SoftConsole running each debugging a different target CPU over separate FlashPro programmer connections.

Note that if more than one FlashPro programmer is attached then SoftConsole debugging will fail with possibly arcane errors. If this happens then ensure that only one FlashPro programmer is attached or else configure the debug launch configuration to explicitly select one of the two more FlashPro programmers that are attached to the PC. To do this:

1. Run the debug sprite on the command line with the -i command line option to inquire about the attached FlashPro programmers – e.g.:

```
C:\Program Files\Microsemi\SoftConsole v3.4\Sourcery-
G++\bin>arm-none-eabi-sprite -i
CodeSourcery ARM Debug Sprite (FlashPro Sprite 1.0-8 +
Actel 1.3.7-M3 + Flash Programming 1.2.2)
flashpro: [jtagclock=<n:93750-4000000>&cpu=<arch>] Actel
FlashPro device
flashpro://usb51451/ - Flash Pro 3B Device
flashpro://usb72412/ - Flash Pro 3B Device
```

2. Edit the debug launch configuration initialization commands to ensure that the debug sprite is invoked with the identifier for the relevant FlashPro device:

```
target remote | "${eclipse_home}/../Sourcery-G++/bin/arm-
none-eabi-sprite" flashpro://usb72412?cpu=Cortex-M3
"${build_loc}"
```

29618 – Installer reports that no FlashPro programmer has been connected even when it has on Windows 64 bit: on Windows 64 bit platforms even if a FlashPro programmer has been or is still connected to the host PC the installer will report that this is not the case. This warning can be safely ignored.

Firmware Catalog sample projects generated from Firmware Catalog Libero Configure Firmware view do not automatically appear in Project Explorer: As mentioned previously in the section about using the Firmware Catalog with SoftConsole, when a sample project is generated from a firmware core into the SoftConsole workspace it does not automatically appear in the SoftConsole IDE Project Explorer. Although the project files are resident in the workspace folder it is still necessary to import the project manually into the workspace.

5.2 Flash Programming/Program Download

1246 – Cannot target a Microsemi CPU in a single JTAG chain with multiple devices: SoftConsole can only target a Microsemi CPU for program download and debugging where it is the only device in a JTAG chain. At the moment SoftConsole debug sprites do not support JTAG bypassing of irrelevant devices or “chained” debugging.

2306 – Writes to memory do not happen unless value different to that displayed: When debugging, poking a value to a memory location in a Memory Monitor view only succeeds if the value to be written is different from the value already displayed for that memory location. This presents a problem with write-only locations which read as zero when you actively want to write zero to this location. Some peripheral registers fall into this category. For example, you may write 0xFF to a register which always reads as 0x00 because of the way in which the hardware is implemented. If you subsequently want to actually write 0x00 to this same register, the write will not occur. The workaround is to initiate the write using a GDB command in the console window. For example:

```
set *((unsigned char *)0x10000000) = 0x00
```

4521: CDT setting to display hex values does not take effect on variables in code window: By default, SoftConsole's Eclipse CDT displays variables, expressions, and registers in decimal values. **Window > Preferences > C/C++ > Debug** allows you to choose to display these values in hexadecimal and other formats. These options control the way values appear in the Variables, Expressions, and Register views. However, they do not affect the way variables and expressions appear in the code editor window when debugging. Values continue to be displayed in decimal in this window.

13218 – sprite should confirm that target memory write actually succeeded: Mismatches between the target memory type (e.g. embedded/external SRAM versus embedded/external NVM/flash), the linker scripts memory map descriptions and the selected debug launch configuration can lead to confusing behaviour and arcane log messages. Always ensure that all of these match up correctly.

14377, 14378, 24053 – Flash programming times can be quite slow in some circumstances: Check the FlashPro device activity LED if SoftConsole seems to stall while downloading a program to flash. The default timeout for actions to perform flash programming is currently rather long; in situations where an incorrect base address was used for the flash memory in question, the session would appear to hang indefinitely. Make sure the address specified in your linker script matches the actual location of the flash memory device in your design.

Flash programming log messages may not appear: When downloading to flash the flash download progress messages may sometimes fail to appear even though download and execution/debugging works properly.

5.3 Cortex-M3 Debugging

19480 – Registers view does not display all Cortex-M3 registers: The Registers view currently only displays the Cortex-M1 register set even for Cortex-M3 so this means that the Cortex-M3 BASEPRI and FAULTMASK registers are not displayed.

21132 – log messages when downloading to ENVM: When downloading programs to ENVM SoftConsole will display a lot of red log messages reflecting the progress of the download. These messages are normal and should not be confused with errors – e.g.:

```
load
Loading section .init, size 0x460 lma 0x60000000
Loading section .text, size 0x800c lma 0x60000460
Loading section .ARM.exidx, size 0x10 lma 0x6000846c
Loading section .data, size 0x530 lma 0x6000847c
arm-none-eabi-sprite: Using host routines for flash programming
arm-none-eabi-sprite: Start of flash programming
arm-none-eabi-sprite: Program 0x60000000 sector [0x0,+0x80) erase
write
arm-none-eabi-sprite: Program 0x60000000 sector [0x80,+0x80)
unchanged
arm-none-eabi-sprite: Program 0x60000000 sector [0x100,+0x80)
unchanged
arm-none-eabi-sprite: Program 0x60000000 sector [0x180,+0x80)
unchanged
arm-none-eabi-sprite: Program 0x60000000 sector [0x200,+0x80)
unchanged
arm-none-eabi-sprite: Program 0x60000000 sector [0x280,+0x80)
unchanged
arm-none-eabi-sprite: Program 0x60000000 sector [0x300,+0x80)
unchanged
arm-none-eabi-sprite: Program 0x60000000 sector [0x380,+0x80)
unchanged
arm-none-eabi-sprite: Program 0x60000000 sector [0x400,+0x80)
erase write
arm-none-eabi-sprite: Program 0x60000000 sector [0x480,+0x80)
erase write
...
arm-none-eabi-sprite: Program 0x60000000 sector [0x8800,+0x80)
erase write
arm-none-eabi-sprite: Program 0x60000000 sector [0x8880,+0x80)
erase write
arm-none-eabi-sprite: Program 0x60000000 sector [0x8900,+0x80)
erase write
arm-none-eabi-sprite: Program 0x60000000 sector [0x8980,+0x80)
erase write
arm-none-eabi-sprite: End of programming
Start address 0x298, load size 35244
Transfer rate: 1 KB/sec, 63 bytes/write.
```

Cortex-M3 debugging requires that linker script defines

__vector_table_vma_base_address symbol: The consolidated debug support for Cortex-M3 targets make uses of an external GDB script to manage differences between the SmartFusion and SmartFusion2 Cortex-M3 targets. In order to facilitate this the script uses a symbol named `__vector_table_vma_base_address` to initialize the target for debugging. This symbol provides the VMA (Virtual Memory Address or runtime memory address) for the base of the program's vector table and is expected to be provided by the linker script used to link the program.

The sample linker scripts provided by the SmartFusion2 MSS CMSIS and Hardware Abstraction Layer firmware core define this symbol. However the sample linker scripts provided by the current released version of the SmartFusion CMSIS-PAL do not. A future update release of this firmware core will address this issue. For now if you are using SoftConsole v3.4 to target SmartFusion Cortex-M3 and get the following error when debugging:

No symbol "__vector_table_vma_base_address" in current context.

then you will need to edit the linker script used to add the require symbol and relink – for example add the highlighted text:

```
SECTIONS
{
  .text :
  {
    CREATE_OBJECT_SYMBOLS
    __text_load = LOADADDR(.text);
    __text_start = .;
    __vector_table_vma_base_address = .;
    *(.isr_vector)
    ...
  }
```

5.4 Core8051s

Addressing different Core8051s memory spaces using the debugger: To access different memory spaces of a Core8051s design from the debugger, you must prefix the actual address with a fixed memory space designator value, as outlined in the following table:

Memory Space	Memory Space Designator Prefix	Prefixed Address Format (xx or xxxx is the actual address in the target memory space)	Example
XDATA	0x00	0x00xxxx	0x00120C
DATA (including SFRs in upper 128 bytes)	0x4000	0x4000xx	0x40004A
CODE	0x80	0x80xxxx	0x8000F3
IDATA	Not supported	Not supported	Not supported

Note: these memory space designator prefixes are only needed when interacting with the target using the debugger – they should not be used in your program code.

Breakpoints in Flash Memory used by the Core8051s: A debug session using a program running from flash memory will only differ in the choice of breakpoints used by the debugger. Software breakpoints are not possible because they depend upon the ability to write a trap instruction (0xA5) at a given location, and flash memory is normally read-only. Instead, the

Core8051s debug sprite now also supports the use of hardware breakpoints if they are available from a particular design.

A debug session will detect if up to four hardware breakpoints are available on the target. If too many hardware breakpoints are requested, the console will show the error

```
c8051-elf-sprite: only 4 hardware breakpoints available
```

(The number **4** above may differ if you have fewer hardware breakpoints in your design.) If this occurs, disable the extra breakpoints which will not yet be reached. When a given breakpoint stops execution, disable its entry under the *Breakpoints* tab and enable the next one which you expect your code to use.

Registers view include values for some pseudo-registers specific to SDCC:

- `spx` – stack pointer in XDATA memory
- `bpx` – frame pointer in XDATA memory
- `bp` – frame pointer in DATA memory

These are used by SDCC when you enable its use of a pseudo stack. In the **Properties** for your project, select **C/C++ Build** and under **SDCC Compiler**, select **Memory Options** and click **Pseudo Stack (--xstack)**.

SDCC and Keil 8051 source code compatibility issues: For details of this issue please refer to the [SoftConsole v2.1 Release Notes](#).

12982 – Step Over sometimes acts like Step Into on lcall

12987 – 8051: issues with accessing APB XDATA memory: Viewing the APB region of XDATA memory is restricted to just the first byte of each 4-byte location, regardless of the APB data width configured in the Core8051s. As a possible workaround, you can observe the `XWBn` and `XRBn` SFRs when APB transactions are occurring.

12991 - 8051: Problems with visibility/access to IDATA upper 128 bytes: The lower 128 bytes share the same physical memory with directly addressed internal data memory (DATA), so they can still be viewed. However, the upper 128 bytes cannot be viewed. Also, the stack uses the IDATA memory space and so cannot be accessed via the debugger if the stack grows into the upper 128 bytes.

13001 – 8051: propagate memory model change details to all component tools: To change the target memory model from the default option of large, you must select it for both the compiler and the linker. If you only change it for the compiler, you will receive errors at link-time. To change the memory model:

1. In the SoftConsole **Project Explorer** select your project, right-click and select **Properties**
2. Choose **C/C++ Build > Settings**
3. Under **SDCC Linker > Memory Options** select the appropriate option from the **Memory Model** dropdown
4. Choose **SDCC Linker > Miscellaneous**
5. Click the + icon in the **Other options** section, and enter `--model-small`, `--model-medium`, or `--model-large` according to the memory model that you wish to use.
Run `C:\Program Files\Microsemi\SoftConsole v3.4\Sourcery-G++\bin\sdcc --help` or refer to the SDCC documentation (e.g. at <http://sdcc.sourceforge.net/>) for more details on the SDCC compiler command line options.

6. Click **OK** to accept the change, then click **Apply** and **OK** on the project's **Properties** dialog box

13003 – 8051: omf2elf doesn't like trailing '/' in the '-i directory' argument: The `c8051-elf-omf2elf` command has a `-i <directory>` option to specify a directory to look in to automatically diagnose the correct combination of OMF and SDCC-generated CDB input files, required to generate an ELF-format binary version of your program. This option will not accept a directory name if it has a trailing '/' character, like `-i ./`. This will be fixed in a future release.

13043 – 8051: register view needs to be improved: Not all of the Special Function Registers (SFRs) are displayed in the *Registers* view while debugging; only the accumulator "a", the "b" register, and the data pointer "dptr" are included with the other more traditional registers. To see other SFRs, you must use the Memory Monitor view, reading DATA memory in the upper 128 bytes (0x80—0xFF, or specifically 0x400080—0x4000FF as used in the view itself).

13045 – 8051: cannot create a debuggable release build executable: There is no way to create an ELF-format debuggable application as a release build for the Core8051s target. The result of a release build is an Intel Hex file (*.ihx), which will need to be appropriately loaded onto your target.

13046 – 8051: need better control over SDCC optimization options: SoftConsole uses a number of SDCC options to disable most optimizations that cause problems with debugging. By default, SDCC compiles all optimizations and there is no flag dedicated to enabling or disabling them all as a group.

13064 – 8051: XDATA local variable can cause C source level debugging problems.

14536 – 8051: Build fails with missing output files if source file containing `main()` and project have the same name: The `c8051-elf-omf2elf` command may fail if a project contains a source file that has the same name as the project. In particular, if you have a project named "test" and a source file named "test.c" containing the definition of `main()`, then you may get the following error:

```
'Invoking: CodeSourcery OMF2ELF Converter'
c8051-elf-omf2elf -c -i .
omf2elf: could not open test: No such file or directory
make: *** [default.elf] Error 1
```

The workaround is to rename the source file so that it has a different name from that of the project.

14756 – Port P2 required by SDCC --xstack option not supported: Core8051s does not support the P2 port required by the SDCC `--xstack` (external stack) compilation option. This option should not be used.

14807 – c8051-elf-omf2elf generates "Detected inconsistent/missing symbol location information" messages: This can happen due to the way in which SDCC generates and links code in some circumstances. In order to obtain more details, run `c8051-elf-omf2elf` with the `-v` (verbose) flag. Right-click the Core8051s project in **Project Explorer**, choose **Properties > C/C++ Build > Settings > CodeSourcery OMF2ELF Converter**, and append `-v` to the **Command** field.

17897 – 8051: build errors with extern functions: Due to a bug in SDCC v2.6.3 if you do:

```
extern void foo() {}
```

SDCC does not make `foo` a global symbol, but if you do:

```
extern void foo();
void foo() {}
```

it does and correctly emits:

```
global _foo
```

17899 – 8051: various CDT options ignored by SDCC: Some project configuration options presented by CDT are not relevant to SDCC and may cause compile errors if used. For example:

- **Properties > Tool Settings > SDCC Compiler > Misc**
 - **Support ANSI program** (*-ansi*)
- **Properties > Tool Settings > SDCC Linker > General**
 - **Do not use standard start files** (*-nostartfiles*)
 - **Do not use default libraries** (*-nodefaultlibs*)
 - **No startup or default libs** (*-nostdlib*)

17900 – 8051: cannot find source files in project when debugging: The combination of SDCC and Eclipse/CDT in SoftConsole do not cope well with source file names that use hyphens or spaces. The workaround is to avoid the use of hyphens or spaces in source file names for Core8051s projects.

17931 – 8051: incremental build with SDCC -v (verbose) enabled gives meaningless error: Enable verbose output in a project using **Project Properties > C/C++ Build > Settings > Tool Settings > SDCC Compiler > Miscellaneous > Verbose** (*-v*). Build the project. The project should build OK and give verbose progress info as it does. When it finishes choose build (just build - not clean and build) again and you get something like this:

```
**** Build of configuration Debug for project test8051 ****
make all
foobar.d:5: *** multiple target patterns. Stop.
```

18435 – 8051: "BFD: Dwarf Error: mangled line number section" errors when building: when building 8051 programs sometimes the following type of errors are displayed:

```
c8051-elf-objdump -h -S 8051blink.elf > "8051blink.lst"
BFD: Dwarf Error: mangled line number section.
BFD: Dwarf Error: mangled line number section.
BFD: Dwarf Error: mangled line number section.
BFD: Dwarf Error: mangled line number section.
'Finished building: 8051blink.lst'
```

18934 – 8051: debug session reports error but none in project: If SDCC sees unreachable code it will issue a warning during compile. However, currently SoftConsole sees all messages from SDCC as compiler errors, and will claim your build failed. This can make attempts to launch a debug session produce a dialog about an error in your workspace. While you can tell it to continue to launch your debug session, as another workaround you can instead make SoftConsole build your project again. This error state will go away because there will be no other commands to be run, which SoftConsole will interpret as meaning the build was successful.

20573 - 8051: PSW SFR may be misreported depending on how it is accessed: access to the PSW SFR via DATA memory space (upper 128 bytes) may report the wrong value compared to accesses via the Registers view or `p $psw` GDB console command.

30301 - 8051: assembler build step missing from project settings: in SoftConsole v3.3 (and SoftConsole v3.2 but not SoftConsole v3.1 and earlier) the SDCC assembler build step is missing from the default Core8051s project tool settings for the Debug configuration/build target. Because of this projects that use assembler (**.s* or **.S*) source files will not link properly. For example a

project that uses the HAL (Hardware Abstraction Layer) firmware core configured for Core8051s large memory model will generate the following errors at link time:

```
'Invoking: SDCC Linker'  
sdcc --debug --noinduction --nooverlay --no-peep --model-large -  
o"test8051.hex" ./main.rel ./hal/Core8051s/SDCC/hal_assert.rel  
./hal/Core8051s/SDCC/hw_reg_access.rel  
./drivers/CoreGPIO/core_gpio.rel  
  
?ASlink-Warning-Undefined Global '_HW_set_32bit_reg' referenced  
by module '___hal_Core8051s_SDCC_hw_reg_access'  
  
?ASlink-Warning-Undefined Global '_HW_set_32bit_reg' referenced  
by module '___drivers_CoreGPIO_core_gpio'  
  
?ASlink-Warning-Undefined Global '_HW_set_16bit_reg' referenced  
by module '___hal_Core8051s_SDCC_hw_reg_access'  
  
?ASlink-Warning-Undefined Global '_HW_set_16bit_reg' referenced  
by module '___drivers_CoreGPIO_core_gpio'  
  
...
```

The workaround for this problem is to manually add the SDCC assembler build step to the project's tool settings for the Debug configuration/build target as follows.

1. Right-click the Core8051s project in SoftConsole's *Project Explorer* and choose **Properties**.
2. Go to **C/C++ Build > Tool Chain Editor**
3. From the **Configuration** dropdown list choose **Debug**.
4. Under **Used tools** click the **Select Tools** button
5. In the **Select tools** dialog select **SDCC Assembler** from the **Available tools** list and click the **Add tool** button to add it to the **Used tools** list.
6. Click **OK** in the **Select tools** dialog.
7. Click **OK** in the project **Properties** dialog.
8. Right-click the Core8051s project in SoftConsole's **Project Explorer** and choose **Clean Project** followed by **Build Project** and the project (including any assembler source files) will now build and link correctly.

6 Documentation

Please refer to the Microsemi (formerly Actel) website (www.actel.com) for more extensive documentation on how to use SoftConsole.

7 System Requirements

7.1 Supported Platforms

- Microsoft Windows 8 Pro or Enterprise 32-bit and 64-bit

- Microsoft Windows 7 Professional 32-bit and 64-bit
- Microsoft Windows Vista Business 32-bit and 64-bit
- Microsoft Windows XP Professional with SP3 32-bit and 64-bit
- SoftConsole may run on other Windows 8/Windows 7/Vista/XP variants but it is not supported on anything other than those listed above.

Note: Administrator privileges are required in order to install SoftConsole. Once installed administrator privileges are not required in order to use SoftConsole.

7.2 Software Environment

In order to use SoftConsole to download and debug programs on a hardware target (for example, a development board), you must first attach a FlashPro programmer device to a USB port on your computer and install the required drivers. Make sure to do this before attempting to download and debug programs on a hardware target. Failing to attach the required programmer results in the following error when attempting to access the hardware target:

```
error: No FlashPro device found
```

Refer to the FlashPro Installation Instructions (http://www.actel.com/download/program_debug/flashpro/default.aspx) for more information about installing the required FlashPro software, hardware, and drivers.

8 Licensing

The individual licenses for the elements that make up SoftConsole are presented during the installation process for your review and acceptance. SoftConsole includes tools covered by the following licenses:

- Eclipse Foundation Software User Agreement
- Eclipse Public License - v 1.0
- CodeSourcery Sourcery G++ Software License Agreement
- GNU GENERAL PUBLIC LICENSE, Version 2, June 1991
- GNU LIBRARY GENERAL PUBLIC LICENSE, Version 2, June 1991
- GNU LESSER GENERAL PUBLIC LICENSE, Version 2.1, February 1999
- expat license
- newlib license
- Oracle Java JRE license
- Oracle Java JRE Third Party Licenses
- GNU GENERAL PUBLIC LICENSE, Version 3, 29 June 2007
- GNU RUNTIME LIBRARY EXCEPTION, Version 3.1, 31 March 2009
- GNU LESSER GENERAL PUBLIC LICENSE, Version 3, 29 June 2007
- Libgloss license
- GNU Free Documentation License, Version 1.3, 3 November 2008
- GNU Free Documentation License, Version 1.2, November 2002