Microsemi SoftConsole v5.1

Release Notes

Table of Contents

Microsemi SoftConsole v5.1

Introduction

These are release notes for Microsemi SoftConsole v5.1.

This document uses `<SoftConsole-install-dir>` as a placeholder for the actual SoftConsole install directory. Where this is mentioned substitute the actual SoftConsole install directory name (e.g. `C:\Microsemi\SoftConsole_v5.1` on Windows or `$HOME/Microsemi_SoftConsole_v5.1` on Linux).

Overview

Key features

- Runs on Windows and Linux.
- Development/debug support for ARM® Cortex®-M and RISC-V CPUs/SoCs in a single package.
- Built using the latest industry standard stock free/open source components and tools for ARM® Cortex®-M and RISC-V firmware development and debugging.
- Support for SmartFusion® and SmartFusion2 ARM Cortex-M3, ARM Cortex-M1 and RISC-V firmware development and debugging.
- Uses OpenOCD for ARM Cortex-M1/3 and RISC-V debugging and SmartFusion/SmartFusion2/Fusion eNVM programming/program download.
- Supports download to and debugging from SmartFusion eSRAM and eNVM, SmartFusion2 eSRAM, eNVM and external RAM (MDDR), Cortex-M1 RAM and Fusion eNVM, and RISC-V RAM.
- Supports FlashPro JTAG programmer for debugging (FlashPro5 on Linux, FlashPro3/4/5 on Windows).
- Supports ARM Cortex-M semi-hosting redirection of standard/file I/O from target board to host debugger.
- Allows users to install arbitrary additional Eclipse plug-ins and features.
- Includes a built-in terminal emulator for connecting to a target board's serial port.

Features not supported

- Compatibility with SoftConsole v3.4 workspaces/projects/debug launch configurations. SoftConsole v3.4 workspaces/projects/debug launch configurations cannot be used with SoftConsole v4. SoftConsole v4 workspaces/projects/debug launch configurations cannot be use with SoftConsole v3.4.
- Debugger driven download of programs to non CFI external parallel flash memories.
- Launching Firmware Catalog from SoftConsole.
- Core8051/Core8051s firmware development and debugging. Use Keil C51 Development Tools with Core8051/Core8051s ISD-51 support.

Quick start guide

1. Read the release notes in full.
2. Follow the installation instructions below for the relevant OS platform.
3. Run SoftConsole from the desktop shortcut or "Start" menu entries created by the installer. This will launch SoftConsole and open the example workspace.
4. To use the example projects in the example workspace on an actual board it is necessary to update the projects to match the target hardware – for example by generating the relevant HAL/CMSIS and firmware drivers from Libero SoC or the Firmware Catalog and copying the generated files into the project.
5. The example projects come with default debug launch configurations for debugging. If necessary modify the settings passed to OpenOCD so that they match the actual target hardware.

6. Use the Microsemi website (https://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support), Firmware Catalog (https://www.microsemi.com/products/fpga-soc/design-resources/design-software/firmware-catalog) and/or the Microsemi github (https://github.com/RISCV-on-Microsemi-FPGA) to obtain example/demo/reference design projects and firmware.

Supported platforms

- Operating systems (32 and 64 bit versions except where noted)

  NOTE: physical machines only recommended/supported - see the Known Issues section for details of issues with virtual machines.

  - Windows
    - 7
    - 8.1
    - 10
  - Linux
    - CentOS and Red Hat Enterprise Linux (RHEL)
      - 6.9
      - 7.3 (64 bit only)
    - Ubuntu
      - 14.04 LTS
      - 16.04 LTS
    - openSUSE
      - LEAP 42.2 (64 bit only)
    - Debian
      - 8.8.0

- CPUs
  - Microsemi SmartFusion2 ARM Cortex-M3
  - Microsemi SmartFusion ARM Cortex-M3
  - Microsemi ARM Cortex-M1 for M1 IGLOO, ProASIC3, ProASIC3L and Fusion FPGAs
  - Microsemi ARM Cortex-M1 for RTG4 and PolarFire FPGAs
  - Microsemi RV32IM RISC-V

- Boards
  - SmartFusion2
    - SmartFusion2 Advanced Development Kit – M2S150-ADV-DEV-KIT
    - SmartFusion2 Security Evaluation Kit – M2S090TS-EVAL-KIT
    - SmartFusion2 Starter Kits – SF2-STARTER-KIT and SF2-484-STARTER-KIT
    - Arrow SF2+ Development Kit (M2S010)
    - Creative Development Board (M2S025) by Future Electronics
  - SmartFusion
    - SmartFusion Evaluation Kit – A2F-EVAL-KIT-2
    - SmartFusion Development Kit – A2F500-DEV-KIT-2
  - Cortex-M1 for M1 FPGAs
    - Fusion Embedded Development Kit – M2AFS-EMBEDDED-KIT-2
  - Cortex-M1 for RTG4 and PolarFire FPGAs and RISC-V RV32IM
    - RTG4 Development Kit (RT4G150)
    - PolarFire Evaluation Kit (MPF300TS)

- JTAG Debug
  - FlashPro3, FlashPro4 and FlashPro5 on Windows
  - FlashPro5 on Linux
  - Other JTAG debug probes supported by OpenOCD may be used but are not specifically tested or supported

- Other software
  - Microsemi Libero SoC

    - Microsemi Libero SoC and Firmware Catalog v11.8
    - Microsemi Libero SoC PolarFire v1.1
- Firmware (minimum required version)
    - SmartFusion2 CMSIS Hardware Abstraction Layer 2.3.105
    - SmartFusion CMSIS-PAL 2.4.102
    - Cortex-M1 CMSIS Hardware Abstraction Layer 2.0.7
    - RISC-V Hardware Abstraction Layer (HAL) 2.0.104

- (DirectCore) Hardware Abstraction Layer 2.3.102

Free/Open source packages

Packages used

Microsemi SoftConsole uses a number of free and/or open source packages. Microsemi acknowledges and thanks those organizations and individual developers who work on these projects and make them to others for reuse under the relevant license conditions.

Oracle Java SE	
Version	8u131
Home page	https://www.oracle.com/java/index.html
Documentation	https://www.oracle.com/java/index.html
License	Oracle Binary Code License Agreement for the Java SE Platform Products and JavaFX http://www.oracle.com/technetwork/java/javase/terms/license/index.html
Notes	Oracle Java SE provides the base Java platform on which Eclipse/CDT and other Eclipse plugins run. Credit/thanks to Oracle.
Eclipse/CDT	
Version	Eclipse 4.4.2 (Luna SR2) + CDT 8.6.0 for Eclipse Luna
Home page	https://eclipse.org/luna/
Documentation	https://eclipse.org/luna/
License	Eclipse Public License v1.0 http://www.eclipse.org/legal/epl-v10.html
Notes	Eclipse/CDT – in conjunction with the GNU ARM Eclipse and Roa Logic Eclipse plugin for RISC-V GNU Toolchain – provide the main SoftConsole GUI Integrated Development Environment. The Windows Eclipse/CDT starter.exe has been modified by Microsemi to allow for graceful termination of OpenOCD or other external executables launched from Eclipse. Credit/thanks to the Eclipse/CDT developer community.
GNU ARM Eclipse Plugins	
Version	V3.4.1-201704251808
Home page	http://gnuarmeclipse.github.io/
Documentation	http://gnuarmeclipse.github.io/
License	Eclipse Public License v1.0 http://gnuarmeclipse.github.io/licenses/plug-ins/
Notes	Only the following GNU ARM Eclipse plugins are used: • GNU ARM C/C++ Cross Compiler: provides specific support for ARM targets by way of custom project properties pages and integration with the back end GNU ARM Embedded Toolchain. • GNU ARM C/C++ OpenOCD Debugging: provides specific support for debugging ARM (and RISC-V) targets using OpenOCD from within the Eclipse environment. Credit/thanks to Liviu Ionescu.
GNU ARM Embedded Toolchain	

Version	Windows: 6-2017-q1-update Linux: 5-2016-q3-update
Home page	https://developer.arm.com/open-source/gnu-toolchain/gnu-rm
Documentation	https://developer.arm.com/open-source/gnu-toolchain/gnu-rm
License	https://launchpad.net/gcc-arm-embedded/5.0/5-2016-q3-update/+download/license.txt
Notes	Provides GCC, GDB, binutils, newlib (including newlib nano) etc. development/debug tools for ARM targets (in particular Cortex-M targets). Details of the specific versions of the individual tools in each release package can be found on the ARM Developer website. Credit/thanks to ARM and the GNU ARM Embedded Toolchain development community.
ARM Cortex Microcontroller Software Interface Standard (CMSIS)	
Version	V4.5
Home page	https://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php
Documentation	http://www.keil.com/pack/doc/CMSIS/General/html/index.html
License	Apache 2.0 License: http://www.keil.com/pack/doc/CMSIS/General/html/index.html#License
Notes	Along with the Microsemi SmartFusion2 CMSIS Hardware Abstraction Layer and SmartFusion CMSIS-PAL firmware packages provides a lightweight hardware abstraction layer on which startup code and firmware drivers can operate. Credit/thanks to ARM.
Roa Logic Eclipse Plugin for RISC-V GNU Toolchain	
Version	v2017.1.0.201705151317
Home page	https://github.com/RoaLogic/riscv_gnu_eclipse
Documentation	https://github.com/RoaLogic/riscv_gnu_eclipse
License	Eclipse Public License v1.0 http://www.eclipse.org/legal/epl-v10.html
Notes	As the GNU ARM C/C++ Cross Compiler does for ARM targets this plugin provides specific support for RISC-V targets by way of custom project properties pages and integration with the back end RISC-V GNU toolchain. Microsemi have made some minor modifications to this plugin to better suit the needs of SoftConsole and Microsemi RISC-V Hardware Abstraction Layer (HAL) users. Credit/thanks to Roa Logic/Richard Herveille.
RISC-V GNU Toolchain	
Version	May 3rd 2017 Toolchain Release (https://github.com/riscv/riscv-gnu-toolchain/releases) based on GCC 7.1 and related binutils, newlib etc.
Home page	https://github.com/riscv/riscv-gnu-toolchain
Documentation	https://github.com/riscv/riscv-gnu-toolchain
License	https://github.com/riscv/riscv-gnu-toolchain/blob/master/LICENSE
Notes	The riscv64-unknown-elf prefixed tools support 32 and 64 bit targets and the following multilibs: https://github.com/riscv/riscv-gcc/blob/riscv-next/gcc/config/riscv/t-elf-multilib Credit/thanks to the RISC-V development community.

OpenOCD	
Version	v0.10.0
Home page	http://openocd.org/
Documentation	http://openocd.org/documentation/
License	GNU General Public License v3 http://www.gnu.org/copyleft/gpl.html
Notes	OpenOCD sits between GDB and the target hardware (JTAG debug probe, target board and CPU) to allow for program download and debug using real hardware. When debugging Eclipse launches GDB and then uses GDB's GDB/MI (Machine Interface) to communicate with the debugger. Meanwhile GDB communicates with OpenOCD using OpenOCD's Remote Serial Protocol interface. GDB debug operations are translated into JTAG operations by OpenOCD which communicates with the target hardware/CPU using JTAG via the relevant JTAG debug probe. OpenOCD also has knowledge of specific CPU target debug frameworks (e.g. ARM CoreSight, RISC-V Debug Module) so that it can communicate with and debug supported CPUs. The base OpenOCD v0.10.0 is supplemented by modifications by • Microsemi – to add support for FlashPro, SmartFusion2/SmartFusion/Fusion envm, finding scripts relative to OpenOCD bin directory, other fixes and enhancements • SiFive (https://www.sifive.com/) – to add support for RISC-V debugging Credit/thanks to the OpenOCD development community and to SiFive.
GNU ARM Eclipse Build Tools (Windows only)	
Version	v2.8-20161122
Home page	http://gnuarmeclipse.github.io/windows-build-tools/download/
Documentation	http://gnuarmeclipse.github.io/windows-build-tools/download/
License	http://gnuarmeclipse.github.io/licenses/tools/
Notes	Credit/thanks to Liviu Ionescu.
Inno Setup (Windows only)	
Version	Inno Setup QuickStart Pack v5.5.9-unicode
Home page	http://www.jrsoftware.org/isdl.php
Documentation	http://www.jrsoftware.org/ishelp/
License	Inno Setup License http://www.jrsoftware.org/files/is/license.txt
Notes	Inno Setup is used to create the SoftConsole installer for Windows. Credit/thanks to Jordan Russell.
InstallJammer (Linux only)	
Version	v2.8-20161122
Home page	http://www.installjammer.com/
Documentation	http://installjammer.com/docs/
License	GNU General Public License with exception http://installjammer.com/docs/
Notes	InstallJammer is used to create the SoftConsole installer for Linux.

	While InstallJammer is no longer supported/actively development and maintained it is still a useful and simple way to create GUI wizard based installers for Linux. Credit/thanks to the InstallJammer development community.
RXTX Java Library and Eclipse Plugins	
Version	v2.1-7r4
Home page	http://rxtx.qbang.org/wiki/index.php/Main_Page
Documentation	http://rxtx.qbang.org/wiki/index.php/Main_Page https://mcuoneclipse.com/2015/04/20/serial-terminal-view-in-eclipse-luna/
License	RXTX License v 2.1 - LGPL v 2.1 + Linking Over Controlled Interface http://users.frii.com/jarvi/rxtx/license.html
Notes	RXTX allows the Eclipse TCM Terminal (Console) View to communicate with serial ports in order to provide built-in serial terminal functionality. Credit/thanks to Trent Jarvi and the RXTX development community.

Installation

Windows

Installing

Refer to the Supported Platforms section for details of which Windows versions are supported.

The installer is a 32 bit executable GUI based program named `Microsemi-SoftConsole-v5.1-Windows-Installer.exe`. It must be run with admin privileges. Run the installer and follow the GUI installer wizard instructions on screen.

If the *FPDrivers – InstallShield Wizard* presents the *Modify/Repair/Remove* page then select *Modify* or *Repair* and continue with the installation.

If, when installing the drivers, Windows warns that *"Windows can't verify the publisher of this driver software"* then please select the *"Install this driver anyway"* option. This may happen if the drivers are not signed for the specific Windows version installed.

There may be a slight pause completing the SoftConsole installation after the FlashPro drivers installer has completed – please be patient.

Linux

Refer to the Supported Platforms section for details of which Linux distributions and versions are supported.

Many of the commands below require `root` privileges using `su`, `sudo` or by logging in as `root`.

Before installing

SoftConsole is a 32 bit application therefore before it can be installed on a 64 bit system or run a number of 32 bit packages/libraries must be installed first.

Ubuntu/Debian 64 bit

```
1. dpkg --add-architecture i386
2. apt-get update
3. apt-get install libgtk2.0-0:i386
4. apt-get install libxtst6:i386
5. apt-get install lib32ncurses5
```

CentOS/Red Hat Enterprise Linux 64 bit

```
1. yum install gtk2.i686
2. yum install libXtst.i686
3. yum install ncurses-libs.i686
```

openSUSE (64 bit)

```
1. zypper install gtk2-tools-32bit
2. zypper install libXtst6-32bit
3. zypper install libncurses5-32bit
4. zypper install libgthread-2_0-0-32bit
```

Notes:

1. Most platforms have the `make` and `xdg-utils` packages installed by default but it is advisable to make sure that these are installed using the relevant package management command for the system in use:

   ```
   <package-management-command> install make
   <package-management-command> install xdg-utils
   ```

2. If, when installing the required 32 bit packages on CentOS/RHEL 64 bit, the following error occurs:

```
Error: Protected multilib versions ...
```

then first update the 64 bit package(s) before attempting to install the 32 bit package again. For example if the error occurs when attempting to install gtk2.i686 then do the following:

```
yum upgrade gtk2
yum install gtk2.i686
```

3. It is recommended that the Linux platform used to run SoftConsole has all available updates installed.

4. It may be possible to install and run SoftConsole on other Linux distributions or versions once the required packages are installed. However some earlier distributions (for example, CentOS/RHEL 5.11) may not work and are not recommended or supported.

Installing

1. The installer is a 32 bit executable GUI based program named `Microsemi-SoftConsole-v5.1-Linux-x86-Installer`.

2. Download the installer and ensure that the execute permission bit is set before attempting to run the installer. If it is not then set it as follows from the command line (the following assumes that the installer has been downloaded to `$HOME/Downloads`):

```
cd ~/Downloads
chmod +x Microsemi-SoftConsole-v5.1-Linux-x86-Installer
```

3. Run the installer:

```
./Microsemi-SoftConsole-v5.1-Linux-x86-Installer
```

4. Follow the installer GUI wizard instructions on screen. If the installer does not appear on screen then double check that all of the required dependent packages/libraries were installed as explained previously.

5. If necessary run the installer in debug mode to diagnose problems with running it:

```
./Microsemi-SoftConsole-v5.1-Linux-x86-Installer --debugconsole
```

After installing

By default USB devices are only accessible with root privileges. In order to debug using SoftConsole and FlashPro5 as a non-root user some additional steps must be taken.

1. The user running SoftConsole must be a member of the `plugdev` group. Many recent Linux distributions create this group by default and add new users to it by default. Where this is not the case (for example some versions of CentOS/RHEL) the `plugdev` group will need to be created and the relevant user account added to it manually. When this has been done it may be necessary to log out and in again or even to reboot the machine for the changes to take effect.

2. Copy the OpenOCD udev rules file and tell the udev substystem to load it. This rules file describes all USB JTAG devices supported by OpenOCD to the system and makes them accessible by users belonging to the `plugdev` group without requiring root privileges.

```
cd ~/Microsemi_SoftConsole_v5.1/openocd/share/openocd/contrib
sudo cp 60-openocd.rules /etc/udev/rules.d
sudo udevadm trigger
```

   In some cases it may be necessary to reboot for the changes to take effect.
   If you previously used SoftConsole v4.x or 5.0 and installed the `99-openocd.rules` file into `/etc/udev/rules.d` then you can delete that file (as it is now redundant) and run `udevadm trigger` again or reboot for the changes to take effect.

   To check that FlashPro5 can be used without root privileges…

3. Connect a FlashPro5 JTAG programmer to the host machine and check that it is visible to the operating system:

```
lsusb

Bus 001 Device 004: ID 1514:2008 Actel
```

   If the FlashPro5 device (ID 1514:2008 (vendor ID 0x1514, product ID 0x2008)) does not appear then double check that the previous instructions were carried out correctly.

4. To the JTAG end of the FlashPro5 connect a suitable board containing a Cortex-M1, SmartFusion or SmartFusion2 Cortex-M3, or RISC-V CPU based SoC design. Power the board on. Make sure that the board is configured for FlashPro JTAG debugging of the target CPU (depending on the board and CPU/SoC in use some board switches/jumpers configuration may be required). Run OpenOCD from the command line to ensure that the debug connection can be established to the target CPU/SoC.

```
cd ~/Microsemi_SoftConsole_v5.1/openocd/bin
export LD_LIBRARY_PATH=`pwd`
./openocd -f board/microsemi-cortex-m1.cfg
```

   OR

```
./openocd -c "set DEVICE M2S090" -f board/microsemi-cortex-m3.cfg
```

OR

```
./openocd -f board/microsemi-riscv.cfg
```

For Cortex-M1 the output should be of the form:

```
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'jtag'
adapter speed: 6000 kHz
microsemi_flashpro tunnel_jtag_via_ujtag off
cortex_m reset_config sysresetreq
trst_only separate trst_push_pull
do_board_reset_init
Info : FlashPro ports available: usb50266
Info : FlashPro port used: usb50266
Info : clock speed 6000 kHz
Info : JTAG tap: FPGA.tap tap/device found: 0x2353a1cf (mfg: 0x0e7
(GateField), part: 0x353a, ver: 0x2)
microsemi_flashpro tunnel_jtag_via_ujtag on
Info : JTAG tap: FPGA.tap disabled
Info : JTAG tap: FPGA.dap enabled
Info : Cortex-M1 IDCODE = 0x4ba00477
Info : FPGA.cpu: hardware has 2 breakpoints, 1 watchpoints
cortex_m auto_bp_type off
```

For Cortex-M3 the output should be of the form:

```
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
M2S090
Info : only one transport option; autoselect 'jtag'
adapter speed: 6000 kHz
cortex_m reset_config sysresetreq
trst_only separate trst_push_pull
do_board_reset_init
Info : FlashPro ports available: S201Z7LB20, E200X3ID7
Info : FlashPro port used: S201Z7LB20
Info : clock speed 6000 kHz
Info : JTAG tap: M2S090.tap tap/device found: 0x1f8071cf (mfg: 0x0e7
(GateField), part: 0xf807, ver: 0x1)
Info : JTAG tap: M2S090.tap disabled
Info : JTAG tap: M2S090.dap enabled
Info : Cortex-M3 IDCODE = 0x4ba00477
Info : M2S090.cpu: hardware has 6 breakpoints, 4 watchpoints
```

For RISC-V the output should be of the form:

```
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
```

```
Info : only one transport option; autoselect 'jtag'
adapter speed: 6000 kHz
microsemi_flashpro tunnel_jtag_via_ujtag off
trst_only separate trst_push_pull
do_board_reset_init
Info : FlashPro ports available: S201Z7LB20, E200X3ID7
Info : FlashPro port used: S201Z7LB20
Info : clock speed 6000 kHz
Info : JTAG tap: FPGA.tap tap/device found: 0x1f8071cf (mfg: 0x0e7
(GateField), part: 0xf807, ver: 0x1)
microsemi_flashpro tunnel_jtag_via_ujtag on
Info : JTAG tap: FPGA.tap disabled
Info : JTAG tap: FPGA.dap enabled
Info : RISC-V IDCODE = 0x10e31913
Info : Examined RISCV core; XLEN=32, misa=0x40902223
halted at 0x80000b60 due to debug interrupt
```

5. Output of the following form or other errors (excluding any documented in the known issues section) indicate a problem in which case double check that all of the previous steps have been carried out correctly and that the target hardware/board is correctly configured for debugging of the target CPU/SoC.

```
Info: FlashPro ports available: none
Info: FlashPro port used: usb
Error: InitializeProgrammer(usb) failed: Can not connect to the programmer
```

Troubleshooting

After performing the steps above SoftConsole should run when launched from the system menu or desktop shortcut. If it does not then the most likely cause is some other missing package/library or configuration. In this case run the following commands and check for any errors that arise. If necessary install any other packages that are missing:

```
cd <SoftConsole-install-dir>/eclipse
./eclipse

cd <SoftConsole-install-dir>/openocd/bin
export LD_LIBRARY_PATH=`pwd`
./openocd -v

cd <SoftConsole-install-dir>/arm-none-eabi-gcc/bin
./arm-none-eabi-gdb --version


cd <SoftConsole-install-dir>/riscv-unknown-elf-gcc/bin
./riscv64-unknown-elf-gdb --version
```

Related Microsemi Tools/Resources

Libero SoC/Firmware Catalog

Use Microsemi Libero SoC v11.8 or later to create hardware designs and to export firmware drivers and example projects.

For PolarFire FPGAs use Microsemi Libero SoC PolarFire v1.1 or later.

The Microsemi Firmware Catalog can be used to generate firmware drivers and example projects for use in SoftConsole v5.1.

Firmware drivers

Hardware Abstraction Layers

The following firmware cores (or later versions if available) must be used and can be generated from Libero SoC or from the Firmware Catalog.

- SmartFusion2 CMSIS Hardware Abstraction Layer 2.3.105
- SmartFusion CMSIS-PAL 2.4.102
- Cortex-M1 CMSIS Hardware Abstraction Layer 2.0.7
- RISC-V Hardware Abstraction Layer (HAL) 2.0.104
- (DirectCore) Hardware Abstraction Layer 2.3.102

Warning:

- If earlier versions of these firmware cores are used then there will be problems compiling, linking and/or debugging.

Peripheral firmware drivers

Use Libero SoC or the Firmware Catalog to generate the latest available peripheral drivers for the target system.

Matching firmware to the target hardware

The firmware used in a SoftConsole project must match the target hardware. For SmartFusion and SmartFusion2 projects Libero SoC generates specific firmware files that must be used in order for the SoftConsole project to match and be compatible with the target hardware

The most convenient way to avoid problems is to ensure that Libero SoC is configured to use the appropriate firmware repositories and the LIbero project is configured to use the latest versions of all firmware drivers (including CMSIS/HAL). Then export the firmware from Libero and import/copy the generated files into the SoftConsole project.

Refer to the Libero SoC and Firmware Catalog documentation for more information about the firmware flows supported by these tools.

Warning:

- Before importing/copying Libero SoC or Firmware Catalog generated firmware files into a SoftConsole project it is advisable to manually delete all `CMSIS`, `hal`, `drivers`, `riscv_hal` and `drivers_config` folders from the SoftConsole project leaving only the project specific custom source files.
- For SmartFusion and SmartFusion2 projects the `drivers_config` folder must be generated/exported from Libero SoC and copied/imported into the SoftConsole project every time that the Libero project is modified to ensure that the SoftConsole project matches the target hardware.
- SoftConsole v3.4 workspaces or projects generated by Libero SoC or the Firmware Catalog are not compatible with SoftConsole v5.1 and should not be used.

FlashPro JTAG programmer

SoftConsole includes OpenOCD which uses a FlashPro JTAG programmer for debug access to the target platform/CPU.

On Windows the FlashPro3/4/5 programmers are supported and the relevant drivers must be installed. On Linux only the FlashPro5 programmer is supported and the post-install configuration steps must be carried out to allow access to the FlashPro5 programmer by a non-root user.

SoftConsole v3.4

SoftConsole v3.4 workspaces, projects and debug launch configurations are not compatible with SoftConsole v5 and should not be used. They will not open or operate correctly. Existing SoftConsole v3.4 workspaces, projects and debug launch configurations must be created anew in SoftConsole v5. However this is not an onerous task and is explained elsewhere in the release notes.

Similarly SoftConsole v5 workspaces, projects and debug launch configurations are not compatible with SoftConsole v3.4.

Workspaces

Example workspace

SoftConsole includes an example workspace which is opened by default when you run SoftConsole. This example workspace is located at:

```
<SoftConsole-install-dir>/extras/workspace.examples
```

This workspace contains a number of simple example projects and debug launch configurations that are ready to use once the relevant projects have been updated to match the target hardware – for example by copying the Libero SoC generated drivers_config folder into the project where applicable. It is also advisable to update these example projects with the relevant CMSIS/HAL and firmware drivers generated from the Firmware Catalog.

It is advisable to make a copy of this example workspace and use the copy for experimentation. Note that the SoftConsole uninstaller will delete some or all of this workspace in which case any changes made may be lost.

Refer to the README.txt for each example project for more information.

Example projects

- fpga-cortex-m1-blinky: LED blinker program for a system containing the encrypted HDL soft core CoreCortexM1 (Microsemi:DirectCore:CoreCortexM1:<version>) in an RTG4 or PolarFire FPGA device.
- m1fpga-cortex-m1-blinky: LED blinker program for a system containing the pre placed and routed CortexM1 (Microsemi:DirectCore:CortexM1Top:<version>) in an M1 variant IGLOO, ProASIC3, ProASIC3L or Fusion FPGA device.
- riscv-interrupt-blinky: interrupt driven LED blinker and UART echo program for a system containing the RISC-V RV32IM soft processor.
- riscv-systick-blinky: timer driven LED blinker and UART echo program for a system containing the RISC-V RV32IM soft processor.
- smartfusion-cortex-m3-blinky: LED blinker program for a SmartFusion Cortex-M3 system.
- smartfusion2-cortex-m3-blinky: LED blinker program for a SmartFusion2 Cortex-M3 system.

Example debug launch configurations

Debug launch configurations for each of the above projects. Remember to ensure that the OpenOCD command lines parameters used in the debug launch configuration (*Debugger tab > Other options*) matches the target hardware/board used. Also remember to configure the target hardware for FlashPro debugging (e.g. `JTAG_SEL` tied high and, if applicable, FlashPro/USB rather than RVI debug access enabled).

Be aware of the differences in debug launch configuration settings between Cortex-M1, SmartFusion Cortex-M3, SmartFusion2 Cortex-M3 and RISC-V targets.

When creating new debug launch configurations for a other systems use the example debug launch configurations as a guide or else copy the one that most closely matches the target system and reconfigure it as needed.

Creating a new workspace

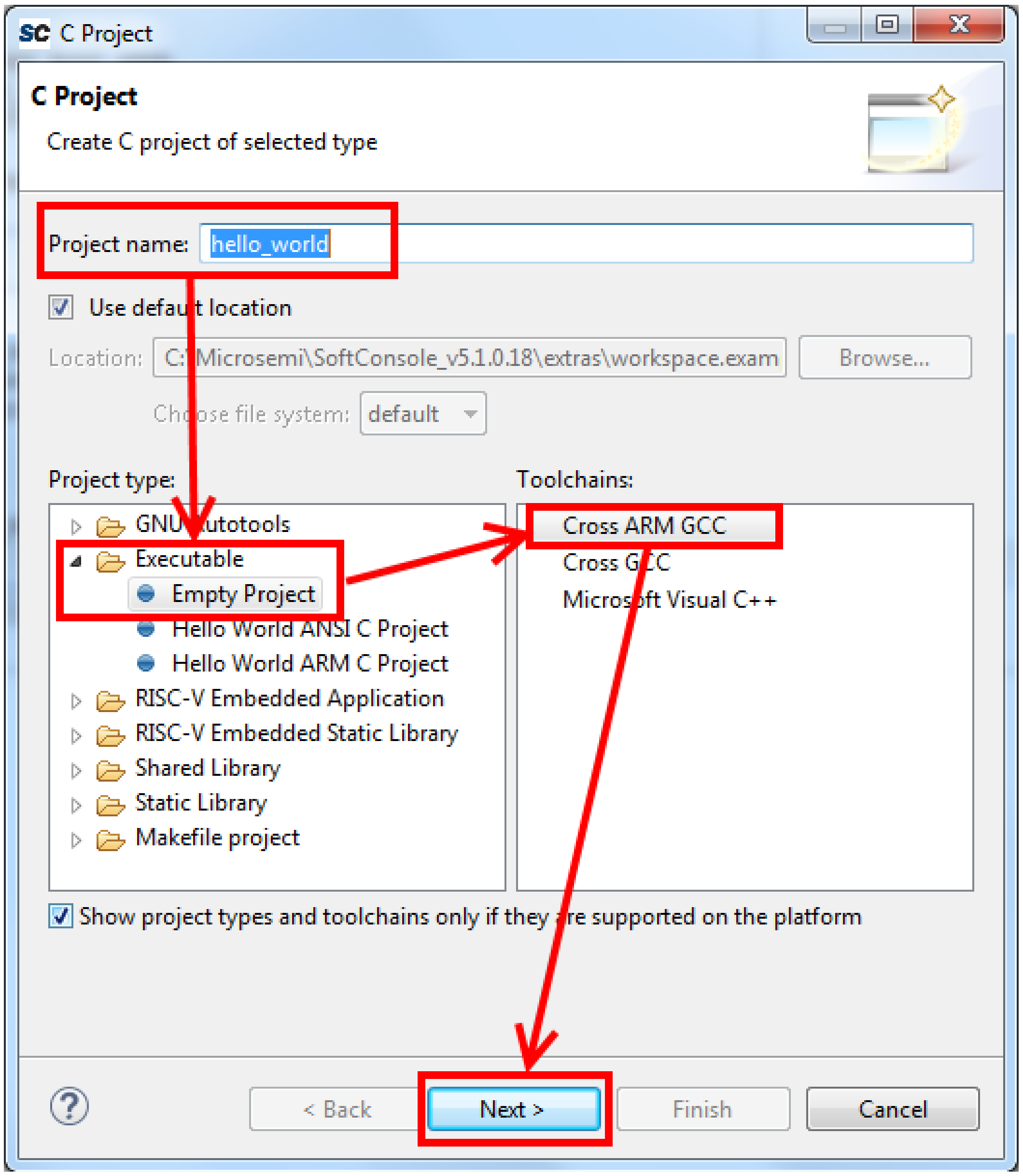
To create a new empty workspace in SoftConsole select *File > Switch Workspace > Other...* and select a folder in which to store the workspace. It is best if a new or empty folder is selected.

Projects

Creating a new Cortex-M project

1. Select *File* > *New* > *C Project* or *C++ Project* depending on the type of project required.

2. In the *C/C++ Project* page of the wizard enter the *Project name*, select *Project type* = *Executable* > *Empty Project* (or *Static Library* > *Empty Project* for a library project), select *Toolchains* = *Cross ARM GCC* and click *Next >*.


SC C Project

C Project

Create C project of selected type

Project name: hello_world

☑ Use default location

Location: C:\Microsemi\SoftConsole_v5.1.0.18\extras\workspace.exam Browse...

Choose file system: default

Project type:
- GNU Autotools
- Executable
  - Empty Project
  - Hello World ANSI C Project
  - Hello World ARM C Project
- RISC-V Embedded Application
- RISC-V Embedded Static Library
- Shared Library
- Static Library
- Makefile project

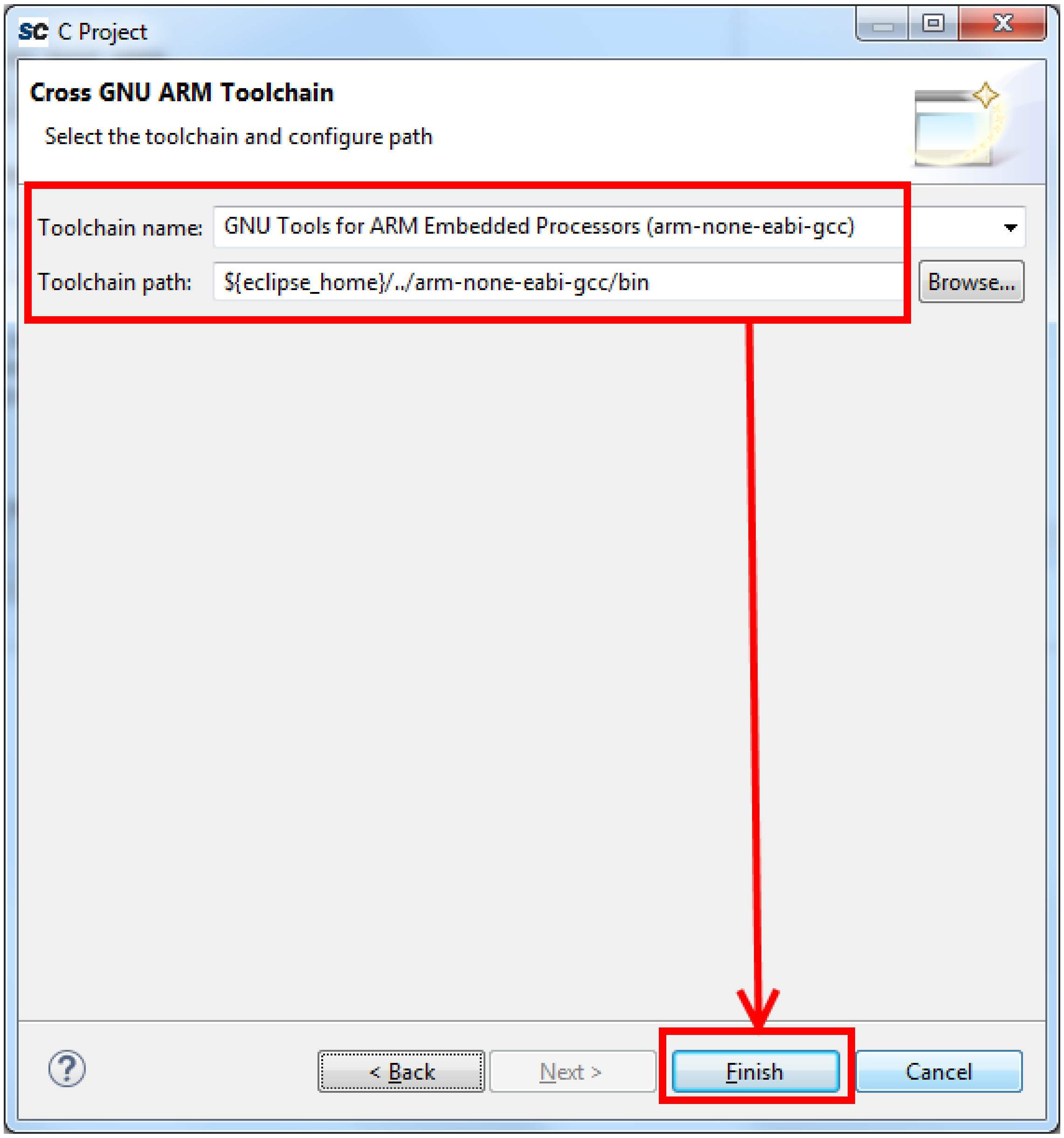
Toolchains:
- Cross ARM GCC
- Cross GCC
- Microsoft Visual C++

☑ Show project types and toolchains only if they are supported on the platform

Figure 1. New Cortex-M Project

3. On the *Select Configurations* page of the wizard click *Next >*.

4. On the *Cross GNU ARM Toolchain* wizard page make sure that *Toolchain name* = *GNU Tools for ARM Embedded Processors (arm-none-eabi-gcc)* and *Toolchain path* = `${eclipse_home}/../arm-none-eabi-gcc/bin`. These are set correctly by default and will remain so unless changed so do not change them. Click *Finish*.


C Project

Cross GNU ARM Toolchain

Select the toolchain and configure path

Toolchain name: GNU Tools for ARM Embedded Processors (arm-none-eabi-gcc)

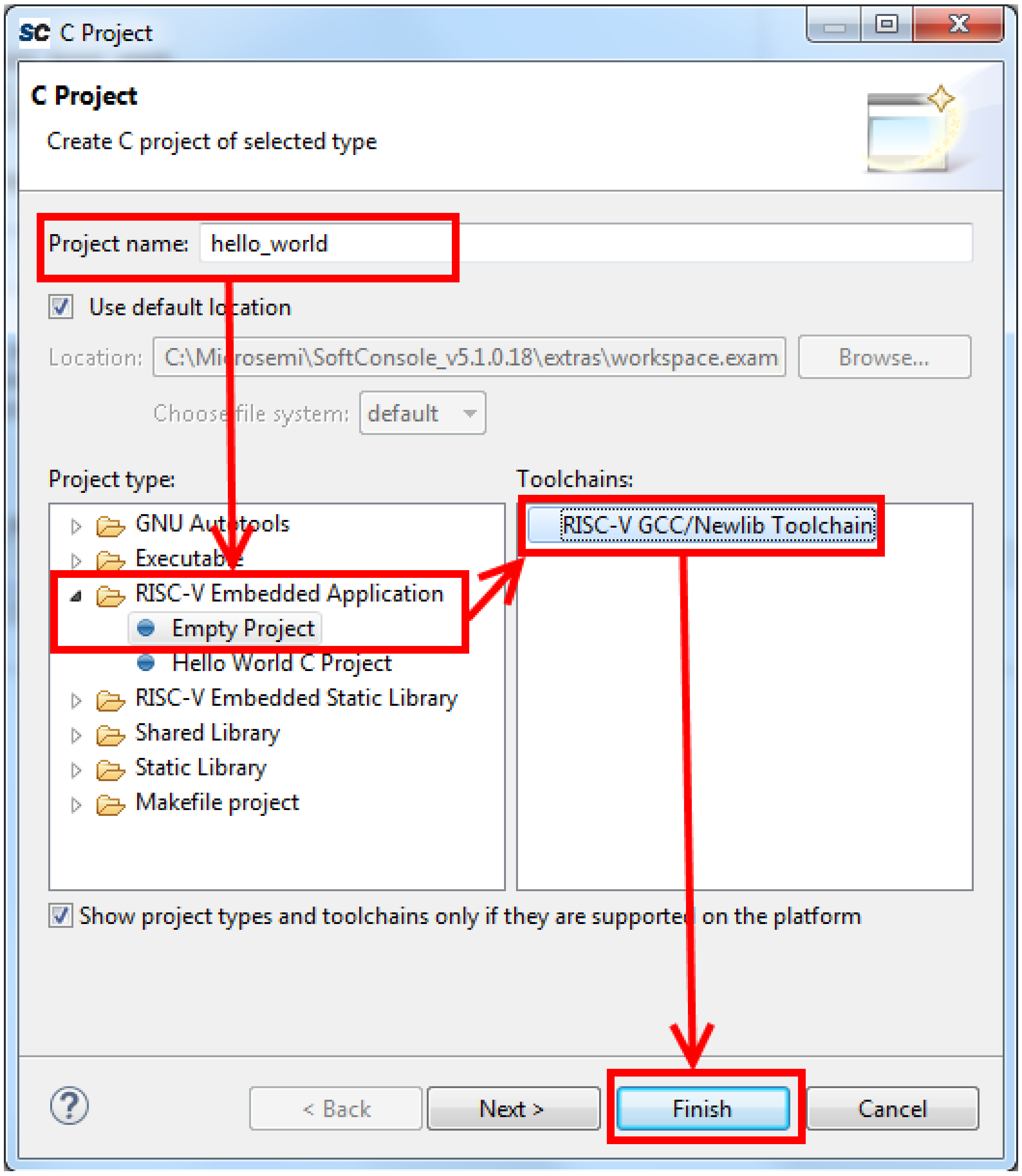
Toolchain path: ${eclipse_home}/../arm-none-eabi-gcc/bin

Figure 2. Cross GNU ARM Toolchain

Creating a new RISC-V project

1. Select *File* > *New* > *C Project* or *C++ Project* depending on the type of project required.

2. In the *C/C++ Project* page of the wizard enter the *Project name*, select *Project type* = *RISC-V Embedded Exextutable* > *Empty Project* (or *RISC-V Embedded Static Library* > *Empty Project* for a library project), select *Toolchains* = *RISC-V GCC/Newlib Toolchain* and click *Next >*.



C Project

C Project

Create C project of selected type

Project name: hello_world

[✓] Use default location

Location: C:\Microsemi\SoftConsole_v5.1.0.18\extras\workspace.exam

Choose file system: default

Project type:
- GNU Autotools
- Executable
- RISC-V Embedded Application
  - Empty Project
  - Hello World C Project
- RISC-V Embedded Static Library
- Shared Library
- Static Library
- Makefile project

Toolchains:
- RISC-V GCC/Newlib Toolchain

[✓] Show project types and toolchains only if they are supported on the platform

Figure 3. New RISC-V Project

3. On the *Select Configurations* page of the wizard click *Finish*.

Project Settings

Some project settings should be modified depending on the target device/CPU.

To modify the project settings right click on the project in the *Project Explorer* and select *Properties* from the context menu. Then navigate to *C/C++ Build* > *Settings*.

Select *Configuration = [All configurations]* to configure settings applicable to all build targets (e.g. *Debug* and *Release*) or else select a specific configuration (e.g. *Debug* or *Release*) to configure settings applicable only to that build target.

Except where noted the settings below can usually be configured for all configurations/build targets.

All targets

Linker Script

It is essential that the appropriate linker script is configured for the project. This will often be one of the example linker scripts bundled with the relevant CMSIS/HAL firmware core which has been generated and imported/copied into the project. For example:

For Cortex-M projects select *Tool Settings* > *Cross ARM C/C++ Linker* > *General* click the *Script files (-T)* > *Add...* button and enter the linker script name into the *Add file path* dialog – e.g.:

- SmartFusion2:
  `"${workspace_loc:/${ProjName}/CMSIS/startup_gcc/debug-in-microsemi-smartfusion2-esram.ld}"`
- SmartFusion:
  `"${workspace_loc:/${ProjName}/CMSIS/startup_gcc/debug-in-actel-smartfusion-envm.ld}"`
- Cortex-M1:
  `"${workspace_loc:/${ProjName}/blinky_linker_config.ld}"`

For RISC-V projects select *Tool Settings* > *RISC-V GCC/Newlib C/C++ Linker* > *General* > *Script file (-T)* and enter the linker script name – e.g.:

- RISC-V:
  `../riscv_hal/microsemi-riscv-ram.ld`

Notes:

- Refer to the relevant CMSIS/HAL documentation for more information about what example linker scripts are available and the circumstances in which they are used.
- In some cases different configurations/build targets will use different linker scripts.

Cortex-M targets

Newlib-Nano

newlib is the standard library bundled with SoftConsole and it is optimized for use in resource/memory constrained bare metal embedded firmware environments. newlib also comes with a "nano" version which is even smaller at the cost of omitting some functionality which may be rarely used in such environments (e.g. the full range of `*printf` formatting options etc.). In many cases it makes sense to use newlib-nano and only switch to the full blown newlib if necessary because using newlib-nano can significantly reduce the compiled and linked programs which use standard library features.

To use newlib-nano check the *Tool Settings* > *Cross ARM C/C++ Linker* > *Miscellaneous* > *Use newlib-nano (--specs=nano.specs)* option.

Create Extended Listing

An extended listing file (e.g. `Debug/<project-name>.lst`) is often useful for understanding the structure and layout of the linked executable.

To enable generation of this file check the *Toolchains > Create extended listing* checkbox.

Preprocessor Defines and Includes

If any preprocessor defines/symbols or includes are needed then they can be specified under *Tool Settings > Cross ARM C/C++ Compiler > Preprocessor > Defined symbols (-D)* and *Tool Settings > Cross ARM C/C++ Compiler > Include paths (-I)* or *Include files (-include)* respectively.

Depending on the target CPU and CMSIS/HAL used additional CMSIS/HAL related include paths may be required. Refer to the relevant CMSIS/HAL documentation for more information.

Optimization Options

Most optimization options can be set at the project "top level" under *Tool Settings > Optimization*.

Other optimization settings, including *Language standard* (which defaults to *GNU ISO C11 (-std=gnu11)* or *GNU ISO 2011 C++ (-std=gnu++11)*), can be specified under *Tool Settings > Cross ARM C/C++ Compiler > Optimization*.

"Fine grained" linking using `-fdata-sections -ffunction-sections` and `-gc-sections` is enabled by default here and also under *Tool Settings > Cross ARM Linker > General > Remove unused sections (-Xlinker --gc-sections)*.

Library Dependencies

Where an application project depends on a static library project this dependency can be configured in the application project's properties so that building the application will ensure that the static library project is also built and up to date if necessary.

Note: for this to work the same configuration/build target (e.g. Debug or Release) must be selected for both projects: e.g. right click on each project and from the context menu select *Build Configurations > Set Active > Debug* or *Release* or any other configuration/build target.

To configure such an application/library project dependency right click on the application project in *Project Explorer* and from the context menu select *Properties* then *Project References* and check the library project(s) on which the application project depends.

Cross ARM GNU Print Size

By default the *Cross ARM GNU Print Size* build step is configured to output size information in "Berkeley" format. The alternative, "SysV" format is often more informative and useful. To change this option right click on the project in *Project Explorer* and from the context menu select *Properties* then *C/C++ Build > Settings > Tool Settings > Cross ARM GNU Print Size > General >* and select *Size format = SysV* instead of *Berkeley*.

Other Options

There are many other options that can be set if needed. Explore the SoftConsole project properties dialog and refer to the relevant GNU/GCC tool documentation for more information on these.

Specifying Options for All Build Configurations

Some project settings can be set once for all built configurations/targets (e.g. Debug and Release). To do this select *Configuration = [All Configurations]* before specifying the relevant options and applying/saving them.

SmartFusion2 Cortex-M3 targets

CMSIS

SmartFusion2 projects require an additional setting in order for the preprocessor to find the toolchain CMSIS header files otherwise compilation will fail to find `core_cm3.h`, `core_cmFunc.h` and/or `core_cmInstr.h`.

Under *Tool Settings > Cross ARM C/C++ Compiler > Miscellaneous* set *Other compiler flags* = `--specs=cmsis.specs`.

Production-Smartfusion2-Relocate-to-External-Ram.ld

For a SmartFusion2 Cortex-M3 program linked using the SmartFusion2 CMSIS Hardware Abstraction Layer example linker script `production-smartfusion2-relocate-to-external-ram.ld` some additional settings must be specified.

When this linker script is used the hex (Intel HEX or Motorola S-record) file generated by SoftConsole is normally used as the input file to a Libero SoC eNVM Data Storage client which is used to program the production firmware into eNVM.

If the following project settings are not configured then the eNVM Data Storage client will reject the hex file as invalid.

Under *Tool Settings > Cross ARM GNU Create Flash Image > General > Other flags* enter `--change-section-lma *-0x60000000`.

This has the effect of "normalising" addresses in the Cortex-M3 memory map view of eNVM (based at 0x60000000) to the more restricted view of memory of the eNVM Data Storage client which only sees eNVM based at 0x00000000.

For more on this and other objcopy options see here: https://sourceware.org/binutils/docs/binutils/objcopy.html.

SmartFusion Cortex-M3 targets

There are no additional settings required for SmartFusion Cortex-M3 projects.

Cortex-M1 targets

Target Processor

The target processor for a project is configured under *Tool Settings > Target Processor > ARM family*. New projects default to having this set to *cortex-m3* to suit SmartFusion and SmartFusion2 Cortex-M3 targets. When targeting Cortex-M1 this must be changed to *cortex-m1*.

CMSIS

Cortex-M1 projects require an additional setting in order for the preprocessor to find the toolchain CMSIS header files otherwise compilation will fail to find certain CMSIS header files.

Under *Tool Settings > Cross ARM C/C++ Compiler > Miscellaneous* set *Other compiler flags* = `--specs=cmsis.specs`.

Adding source files to a project

Once the project has been created the required source files should be added.

In most cases the best way to do this is to use Libero SoC to select the relevant firmware cores (including CMSIS/HAL, SmartFusion/SmartFusion2 MSS peripheral drivers, DirectCore drivers etc.), generate these, export the firmware files and then import or copy them into the SoftConsole project.

In fact for SmartFusion and SmartFusion2 is it essential that at least the `drivers_config` folder is generated by/exported from Libero SoC and imported/copied into the SoftConsole project every time that the hardware project is changed. This is because the files in this folder contain information about the target platform that is essential to the correct functioning on firmware on that hardware platform.

It is also possible to generate specific firmware cores/drivers from the Firmware Catalog and then import/copy them into the SoftConsole project.

Refer to the Libero SoC and Firmware Catalog tools and documentation for more information on generating/exporting firmware cores from these tools.

Warning: remember that any SoftConsole v3.4 workspaces or projects generated by Libero SoC or the Firmware Catalog cannot be used with SoftConsole v5.

When importing/copying firmware files generated by/exported from Libero SoC or the Firmware Catalog it is safest to first manually delete all relevant folders from the SoftConsole project (e.g. `CMSIS`, `hal`, `drivers`, `drivers_config`) and retain only the custom source files created for the project itself.

Firmware folders/files can be copied by dragging and dropping from a file manager on Windows or Linux or by using the SoftConsole import facility. Right click on the project in the *Project Explorer* and from the context menu select *Import...* then select *General > File System* and click *Next >*. Browse to and select the directory from which the firmware files are to be imported (e.g. the `firmware` directory below a Libero SoC project directory), select the required folders/files and click *Finish* to import the files.

Building a project

Once a project has been correctly configured and populated with the required firmware it can be built.

Select/click on the project in the *Project Explorer* and from the application menu select *Project* > *Build Configurations* > *Set Active* and select the required configuration/build target – usually one of *Debug* or *Release*.

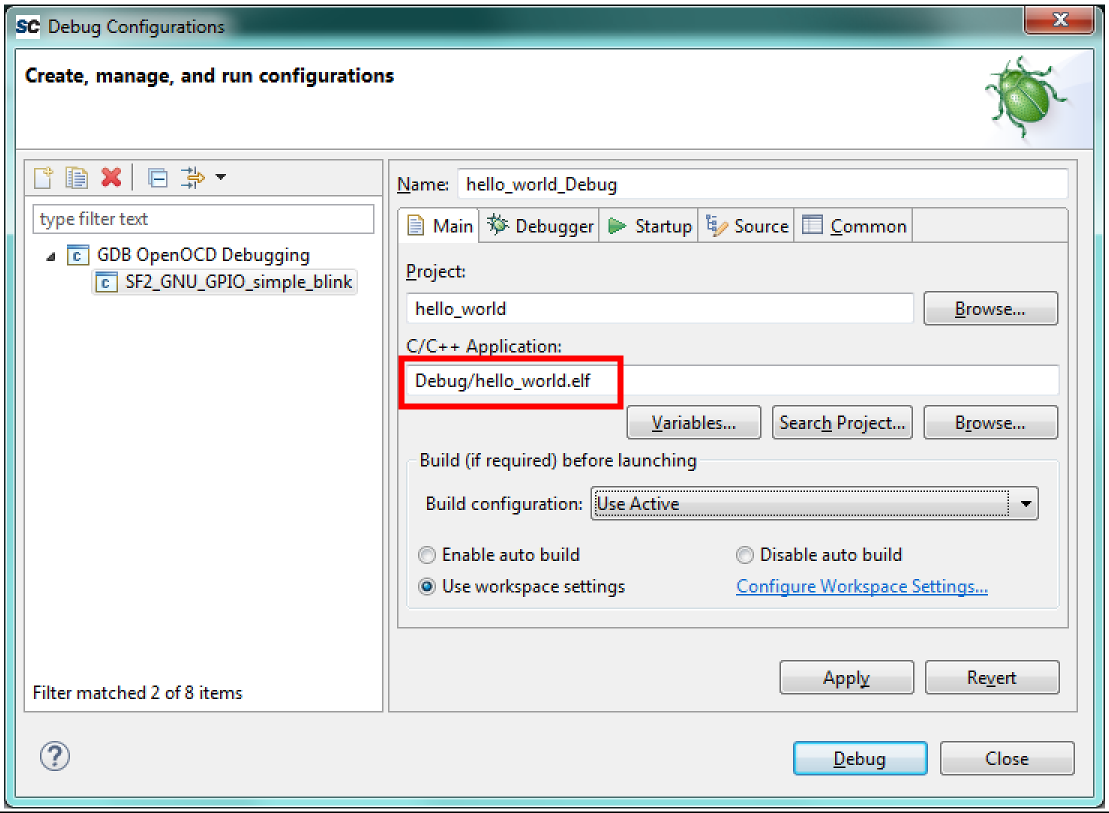
With the project still selected in the *Project Explorer* select *Project* > *Build Project*. The results of the build process can be viewed in the *Console* view and the *Problems* view if there are any problems (e.g. errors or warnings).

Debugging

Debug launch configurations

In order to debug a program a debug launch configuration must be created. Most of the default settings for a debug launch configuration can be left as they are but a few needs to be manually configured.

1. Select the project in the *Project Explorer* and from the SoftConsole application menu select *Run > Debug Configurations...*
2. In the *Debug Configurations* dialog select *GDB OpenOCD Debugging* and click on the *New launch configuration* button which will create a new debug launch configuration for the previously selected project.
3. On the *Main* tab ensure that the *C/C++ Application* field contains the correct executable name. Note that using forward slashes in paths here aids portability of projects and debug launch configurations between Windows and Linux:



Name: hello_world_Debug

Project: hello_world

C/C++ Application: Debug/hello_world.elf

Build (if required) before launching

Build configuration: Use Active

Enable auto build [✓ Use workspace settings] Disable auto build

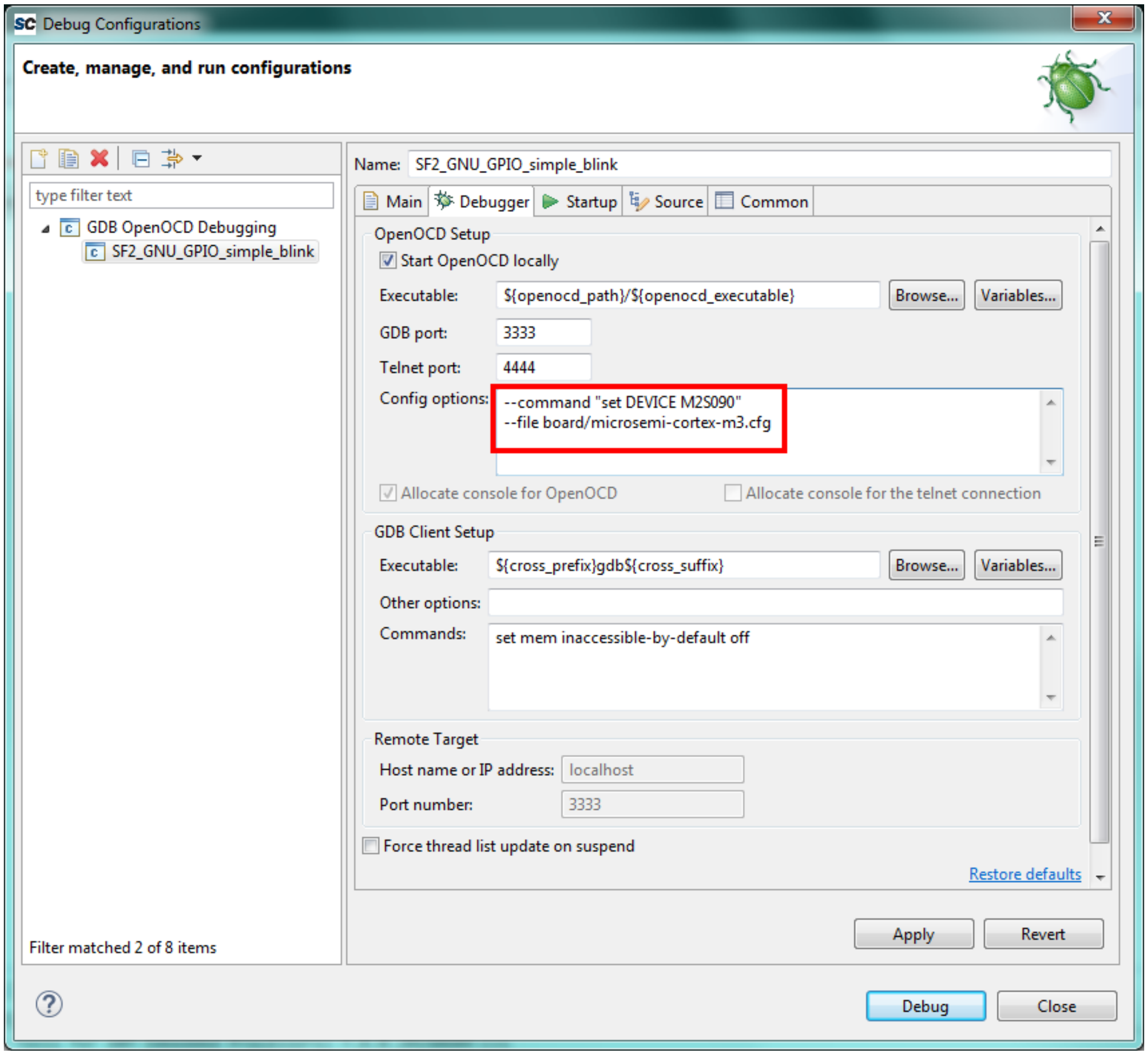
Configure Workspace Settings...

Figure 4. Debug launch configuration Main tab

4. On the *Debugger* tab it is critical that the *Config options* field contains the correct command line options/script to be passed to OpenOCD. The example settings here work for SmartFusion or SmartFusion2 targets where the program uses only eSRAM and/or eNVM – as long as the `DEVICE` setting is modified to match the actual target device (SmartFusion A2FXXX or SmartFusion2 M2SXXX where XXX is the three digit device size designator). Further details about these options are provided elsewhere in this documentation.

   For a Cortex-M target the *Config options* should be:

   ```
   --file board/microsemi-cortex-m1.cfg
   ```



Debug Configurations

Create, manage, and run configurations

type filter text

- GDB OpenOCD Debugging
  - SF2_GNU_GPIO_simple_blink

Filter matched 2 of 8 items

Name: SF2_GNU_GPIO_simple_blink

Main | Debugger | Startup | Source | Common

OpenOCD Setup

☑ Start OpenOCD locally

Executable: ${openocd_path}/${openocd_executable}

GDB port: 3333

Telnet port: 4444

Config options:

```
--command "set DEVICE M2S090"
--file board/microsemi-cortex-m3.cfg
```

☑ Allocate console for OpenOCD ☐ Allocate console for the telnet connection

GDB Client Setup

Executable: ${cross_prefix}gdb${cross_suffix}

Other options:

Commands: set mem inaccessible-by-default off

Remote Target

Host name or IP address: localhost

Port number: 3333

☐ Force thread list update on suspend

Figure 5. Debug launch configuration Debugger tab for Cortex-M3

5. For a RISC-V target the Debugger tab settings must be different – specifically:

   OpenOCD Setup > Config options:
   ```
   --file board/microsemi-riscv.cfg
   ```

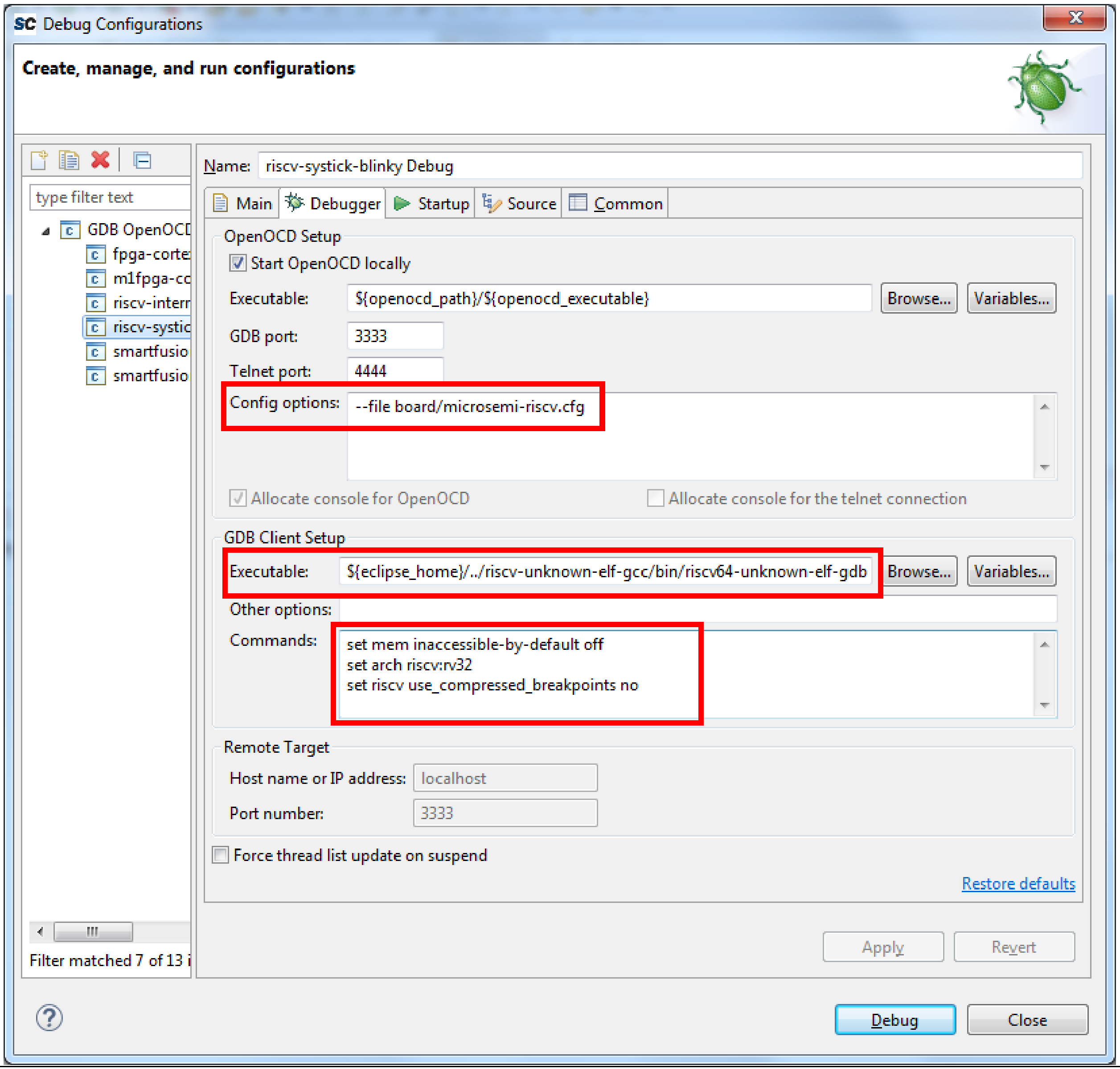
GDB Client Setup > Executable:
`${eclipse_home}/../riscv-unknown-elf-gcc/bin/riscv64-unknown-elf-gdb`

GDB Client Setup > Commands:

```
set mem inaccessible-by-default off
set arch riscv:rv32
set riscv use_compressed_breakpoints no
```

Notes:

Use `set arch riscv:rv64` if targeting a RISC-V RV64 processor.
Omit `set riscv use_compressed_breakpoints no` if targeting a RISC-V processor that implements the C (compressed instructions) extension.



Name: riscv-systick-blinky Debug

OpenOCD Setup
[✓] Start OpenOCD locally
Executable: ${openocd_path}/${openocd_executable}
GDB port: 3333
Telnet port: 4444
Config options: --file board/microsemi-riscv.cfg
[✓] Allocate console for OpenOCD
[ ] Allocate console for the telnet connection

GDB Client Setup
Executable: ${eclipse_home}/../riscv-unknown-elf-gcc/bin/riscv64-unknown-elf-gdb
Other options:
Commands:

```
set mem inaccessible-by-default off
set arch riscv:rv32
set riscv use_compressed_breakpoints no
```

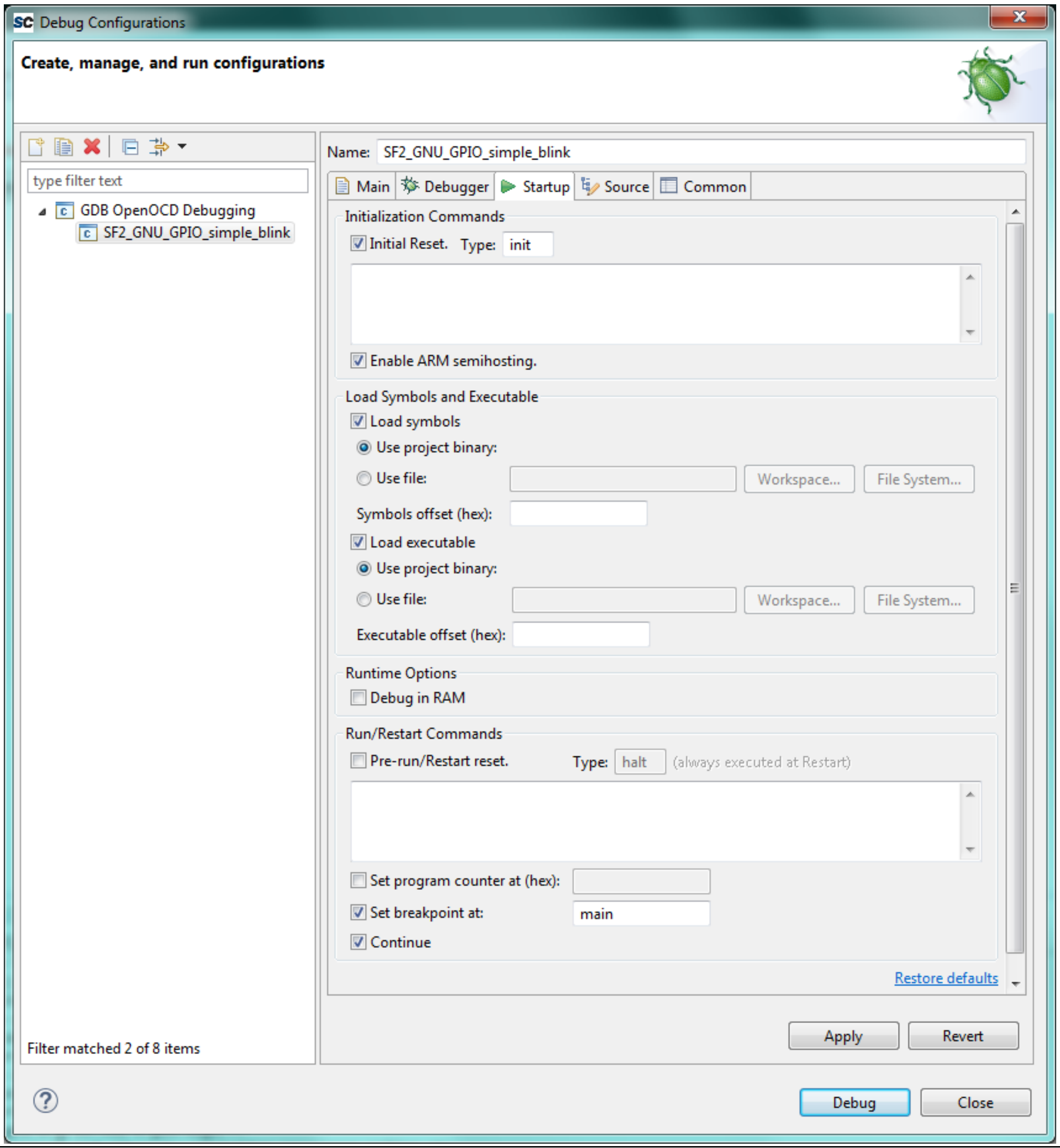
Remote Target
Host name or IP address: localhost
Port number: 3333
[ ] Force thread list update on suspend

Figure 6. Debug launch configuration Debugger tab for RISC-V

6. On the *Startup* tab the default settings should be configured as shown below and these are the default settings so do not change them unless absolutely necessary and you understand what effect these changes will have.

   Initialization Commands > Initial Reset must be checked and *Type* set to *init*. *Enable ARM semihosting* can be enabled whether or not semihosting will be used – it should be disabled for RISC-V targets since they do not support semihosting.

   Load symbols/executable should be configured as shown. *Runtime Options > Debug in RAM* should always be disabled – even when targeting embedded or external RAM. *Run/Restart Commands > Pre-run/Restart reset* must be disabled. *Set breakpoint at main* and *Continue* should normally be checked although can be modified if, for example, an initial breakpoint somewhere other than `main()` is required or startup code executed before `main()` needs to be debugged.



Name: SF2_GNU_GPIO_simple_blink

Initialization Commands
[✓] Initial Reset. Type: init
[✓] Enable ARM semihosting.

Load Symbols and Executable
[✓] Load symbols
(●) Use project binary:
( ) Use file:
Symbols offset (hex):
[✓] Load executable
(●) Use project binary:
( ) Use file:
Executable offset (hex):

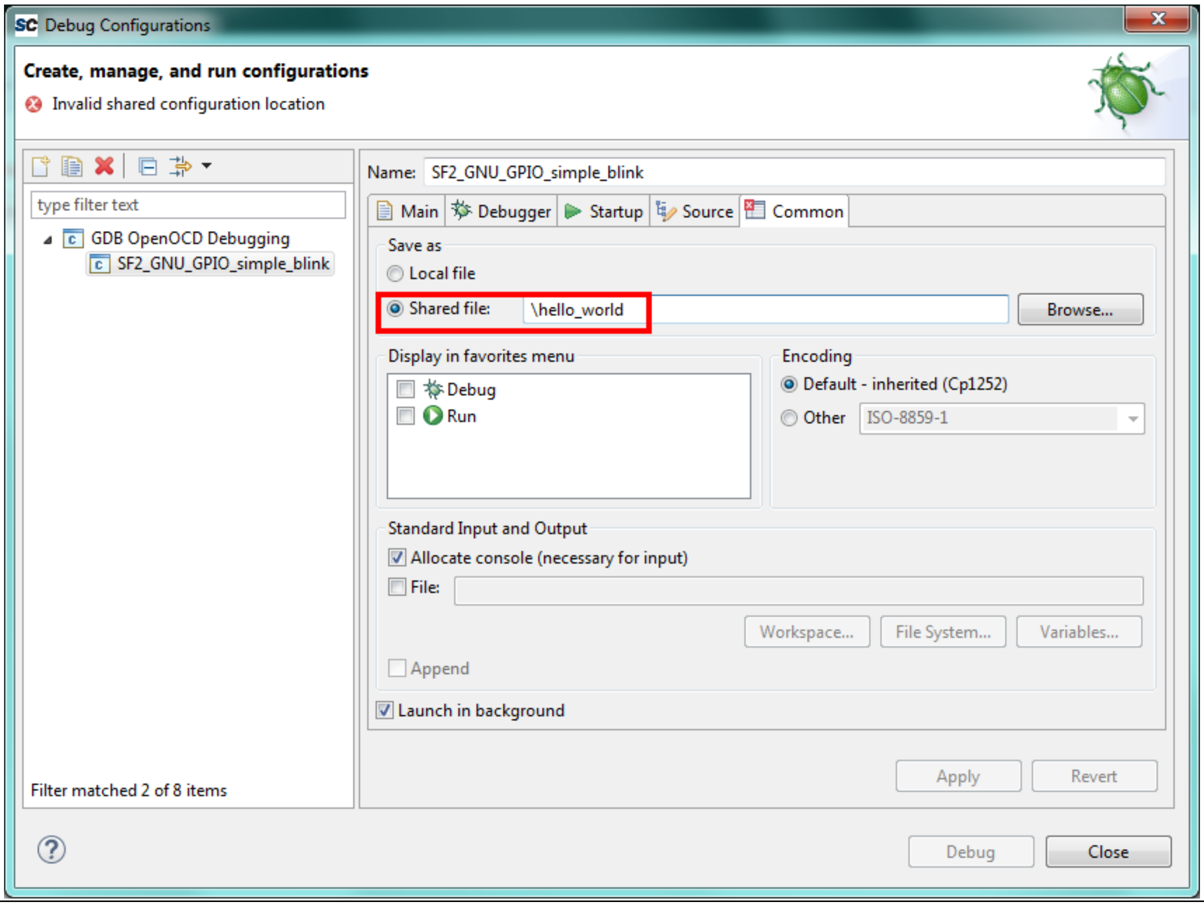
Runtime Options
[ ] Debug in RAM

Run/Restart Commands
[ ] Pre-run/Restart reset. Type: halt (always executed at Restart)
[ ] Set program counter at (hex):
[✓] Set breakpoint at: main
[✓] Continue

Figure 7. Debug launch configuration Startup tab

7. On the *Common* tab the *Save as* > *Local file* option is selected by default. This causes the debug launch configuration to be saved into the workspace. However if the *Shared file* option is selected (the default name can be accepted) then the debug launch configuration instead gets saved into the project which aids portability as it means that the debug launch configuration moves in tandem with the project (e.g. when copying or exporting/importing the project).

`


Debug Configurations

Create, manage, and run configurations

Invalid shared configuration location

Name: SF2_GNU_GPIO_simple_blink

GDB OpenOCD Debugging
SF2_GNU_GPIO_simple_blink

Save as: Local file [✓ Shared file: \hello_world]

Display in favorites menu: ☐ Debug ☐ Run

Encoding: [✓ Default - inherited (Cp1252)] Other ISO-8859-1

Standard Input and Output
☑ Allocate console (necessary for input)
☐ File:
☐ Append

☑ Launch in background

Filter matched 2 of 8 items

Figure 8. Debug launch configuration Common tab

OpenOCD command line options and scripts

As explained above, it is important that the correct command line options/scripts are passed to OpenOCD via the *Debugger > Config options* setting in the debug launch configuration. This section explains these settings.

Note:

- All `--command ...` settings mentioned below must be placed before the `--file ...` setting.
- Commands can be specified using `--command ...` or `-c ....`
- Multiple commands can be specified individually

```
--command "set DEVICE M2S090" --command "set JTAG_KHZ 1000"
```

or together separated by semi-colons

```
--command "set DEVICE M2S090; set JTAG_KHZ 1000"
```

SmartFusion/SmartFusion2 DEVICE

For SmartFusion and SmartFusion2 the target device must be specified using `--command "set DEVICE <devicename>"`.

For SmartFusion the target device must be set using `--command "set DEVICE A2FXXX"` where XXX is one of 060, 200 or 500.

For SmartFusion2 the target device must be set using `--command "set DEVICE M2SXXX"` where XXX is one of 005, 010, 025, 050, 060, 090 or 150.

Board scripts

The board script describes the relevant aspects of the target hardware to OpenOCD. A number of example scripts are provided and are stored in `<SoftConsole-install-dir>/openocd/share/openocd/scripts`. The following list enumerates these and outlines the context in which each of them can be used. Remember that the target device must also be correctly specified in the debug launch configuration.

- SmartFusion/SmartFusion2 Cortex-M3
  - `board/microsemi-cortex-m3.cfg`: for SmartFusion or SmartFusion2 programs that target only eSRAM or eNVM.

- SmartFusion2 Cortex-M3 only
  - `board/microsemi-smartfusion2-eval-or-starter-kit-ddr.cfg`: an example script supporting a specific SmartFusion2 MDDR configuration on the SmartFusion2 Evaluation Kit, Security Evaluation Kit or either of the Starter Kit boards. For use when downloading to/debugging from MDDR.
  - `board/microsemi-smartfusion2-dev-kit-ddr.cfg`: an example script supporting a specific SmartFusion2 MDDR configuration on the SmartFusion2 Development Kit or Advanced Development Kit boards. For use when downloading to/debugging from MDDR.
  - `board/microsemi-smartfusion2-dev-kit-ddr-ecc.cfg`: an example script supporting a specific SmartFusion2 MDDR configuration with ECC enabled on the SmartFusion2 Development Kit or Advanced Development Kit boards. For use when downloading to/debugging from MDDR with ECC enabled.

- Cortex-M1
  - `board/microsemi-cortex-m1.cfg`: for targeting Cortex-M1. Explained in the next section.

- RISC-V
  - `board/microsemi-riscv.cfg`: for targeting RISC-V.

Note: For more information about SmartFusion2 MDDR external RAM support see elsewhere in this document and also in the `<SoftConsole-install-dir>/extras/smartfusion2-mddr` folder in the SoftConsole installation.

The following outlines the normal correlation between the linker script used to link the program and the OpenOCD board script used for debugging:

SmartFusion2 CMSIS Hardware Abstraction Layer	
Linker script	**OpenOCD board script**
`debug-in-microsemi-smartfusion2-esram.ld` `debug-in-microsemi-smartfusion2-envm.ld`	`board/microsemi-cortex-m3.cfg`
`debug-in-microsemi-smartfusion2-external-ram.ld`	`board/microsemi-smartfusion2-eval-or-starter-kit-ddr.cfg` `board/microsemi-smartfusion2-dev-kit-ddr.cfg` `board/microsemi-smartfusion2-dev-kit-ddr-ecc.cfg`
`production-smartfusion2-execute-in-place.ld` `production-smartfusion2-relocate-to-external-ram.ld`	Not applicable – not for interactive debugging
SmartFusion CMSIS-PAL	
Linker script	**OpenOCD board script**
`debug-in-actel-smartfusion-esram.ld` `debug-in-actel-smartfusion-envm.ld`	`board/microsemi-cortex-m3.cfg`
`debug-in-external-ram.ld`	Not applicable – not yet supported
`production-execute-in-place.ld` `production-relocate-executable.ld`	Not applicable – production flow, not for interactive debugging
Hardware Abstraction Layer (Cortex-M1/DirectCore)	
Linker script	**OpenOCD board script**
`ram-debug.ld`	`board/microsemi-cortex-m1.cfg`
`boot-from-intel-flash.ld` `boot-from-nvm.ld`	Not applicable – production flow, not for interactive debugging
`run-from-nvm.ld` `run-from-intel-flash.ld`	Not applicable – not yet supported
Cortex-M1 CMSIS Hardware Abstraction Layer	
Linker script	**OpenOCD board script**
Refer to the Cortex-M1 CMSIS HAL documentation	`board/microsemi-cortex-m1.cfg`
RISC-V Hardware Abstraction Layer (HAL)	
Linker script	**OpenOCD board script**
Refer to the RISC-V HAL documentation	`board/microsemi-riscv.cfg`

Cortex-M1 Board Script

Use the `board/microsemi-cortex-m1.cfg` board script when targeting a Cortex-M1 based system on chip.

Unlike SmartFusion/SmartFusion2 when targeting Cortex-M1 `--command "set DEVICE ..."` is not needed.

If the Cortex-M1 system includes flash memory then the `board/microsemi-cortex-m1.cfg` board script needs to be modified (or copied and modified) to add this.

The Cortex-M1 can be configured to allow debugging using FlashPro “indirectly” via the FPGA’s UJTAG block or “directly” via general I/O pins carrying the JTAG signals. The board script assumes the former (UJTAG) by default. To override this and select “direct” debugging add the following:

```
--command "set FPGA_TAP N"
```

FlashPro JTAG speed

The SoftConsole OpenOCD scripts use a default JTAG clock speed of 6MHz. If this needs to be overridden then it can be specified (in kHz) alongside the target device – e.g. to use 1MHz (1000kHz):

```
--command "set DEVICE M2S090; set JTAG_KHZ 1000"
```

or

```
--command "set DEVICE M2S090" --command "set JTAG_KHZ 1000"
```

Warning: do not change the JTAG clock speed unless absolutely necessary and only if you understand the implications and possible pitfalls of doing so.

Other OpenOCD options

In some cases where OpenOCD debugging does not work as expected it may be useful to add the `--debug n` (where `n` is a debug level between 0 and 3) to the debug launch configuration.

See also the OpenOCD User's Guide for other OpenOCD options and commands: http://openocd.org/documentation/.

SoftConsole OpenOCD script parameters

A number of parameters can be used to configure/control how the SoftConsole OpenOCD scripts operate.

Refer to the comments in the example scripts for more details.

- `<SoftConsole-install-dir>/openocd/share/openocd/scripts/interface/microsemi-flashpro.cfg`
- `<SoftConsole-install-dir>/openocd/share/openocd/scripts/target/microsemi-cortex-m1.cfg`
- `<SoftConsole-install-dir>/openocd/share/openocd/scripts/target/microsemi-cortex-m3.cfg`
- `<SoftConsole-install-dir>/openocd/share/openocd/scripts/target/microsemi-riscv.cfg`

Board configuration for FlashPro debugging

Debugging a Cortex-M3 target with the FlashPro JTAG programmer requires that JTAG_SEL is tied high and, where applicable, FlashPro/USB rather than RVI debug access is enabled.

If JTAG_SEL is not configured correctly then debugging will not work.

Using a debug session

Launching a debug session

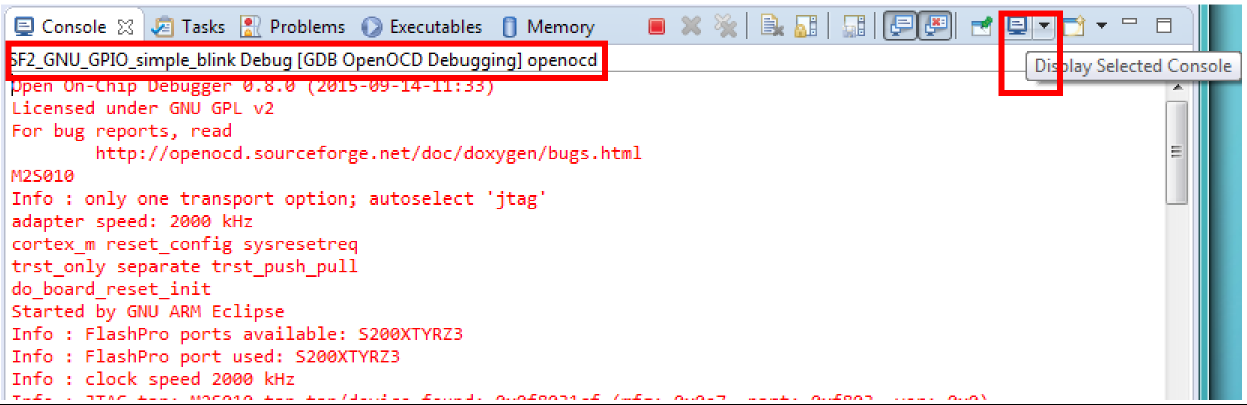
Select the project in the *Project Explorer*, right click on it and from the context menu select *Debug As > Debug Configurations*, select the relevant debug launch configuration and click *Debug*.

Memory Monitor

The default Memory Monitor view rendering is *Hex* which may render values in big-endian rather than little-endian form. If this is the case then switch to *Traditional* or *Hex Integer* rendering which renders values properly as little-endian.

Console view

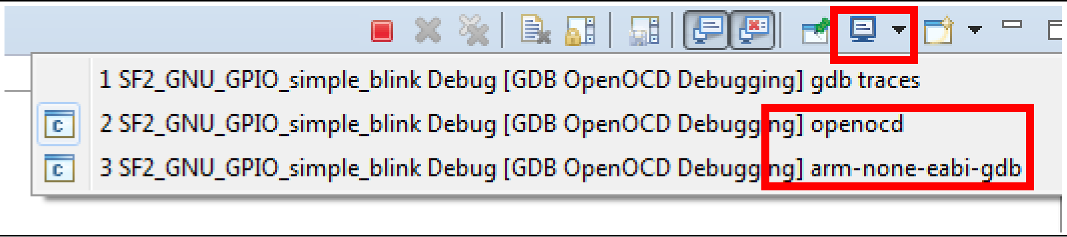
During a debug session SoftConsole can display a number of different consoles in the *Console* view. By default the OpenOCD console is displayed showing OpenOCD output:



```
SF2_GNU_GPIO_simple_blink Debug [GDB OpenOCD Debugging] openocd
Open On-Chip Debugger 0.8.0 (2015-09-14-11:33)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.sourceforge.net/doc/doxygen/bugs.html
M2S010
Info : only one transport option; autoselect 'jtag'
adapter speed: 2000 kHz
cortex_m reset_config sysresetreq
trst_only separate trst_push_pull
do_board_reset_init
Started by GNU ARM Eclipse
Info : FlashPro ports available: S200XTYRZ3
Info : FlashPro port used: S200XTYRZ3
Info : clock speed 2000 kHz
```

Figure 9. Debug session – OpenOCD console view

The highlighted *Display Selected Console* toolbar button allows different consoles to be selected:



```
1 SF2_GNU_GPIO_simple_blink Debug [GDB OpenOCD Debugging] gdb traces
2 SF2_GNU_GPIO_simple_blink Debug [GDB OpenOCD Debugging] openocd
3 SF2_GNU_GPIO_simple_blink Debug [GDB OpenOCD Debugging] arm-none-eabi-gdb
```

Figure 10. Debug session – selecting a specific console view

The *openocd* and *arm-none-eabi-gdb* consoles are usually the ones of most interest. If semihosting is used the I/O is done via the GDB console. The *arm-none-eabi-gdb* console must be the active console in order to manually enter GDB commands.

Built-in serial terminal view

SoftConsole includes a built-in serial terminal view which obviates the need to run a separate serial terminal emulator when connecting to a target board using a UART. The plug-ins used to implement this view are pre-installed. Refer to this blog post for information on how to show and configure the terminal view (but skip the parts dealing with plug-in installation as this is already done):

http://mcuoneclipse.com/2015/04/20/serial-terminal-view-in-eclipse-luna/

Debug using a specific FlashPro programmer

By default SoftConsole will debug using the first FlashPro5 programmer that it detects. If there is no FlashPro5 connected then it will use the first FlashPro3/4 that it detects.

When there is only one FlashPro programmer connected this is fine. In some cases more than one FlashPro programmer will be connected in which case SoftConsole needs to be told which one to use for debugging.

A specific example of this is when using the M2S090 Security Evaluation Kit board. On this board J5 is the FlashPro connector normally used for FlashPro programming of the FPGA and SoftConsole debugging. However J18 is also an on-board SPI only FlashPro5 programmer which can be used for programming the FPGA but cannot be used for SoftConsole debugging. J18 is also used for access to serial ports on the target design.

In this case if both J5 and J18 are connected to the host computer on which SoftConsole is running then SoftConsole needs to be told to use the former for debugging.

When OpenOCD runs it lists the FlashPro programmers that it finds and indicates which one it uses by default – e.g:

```
Open On-Chip Debugger 0.8.0 (2015-09-14-11:33)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.sourceforge.net/doc/doxygen/bugs.html
M2S010
Info : only one transport option; autoselect 'jtag'
adapter speed: 2000 kHz
cortex_m reset_config sysresetreq
trst_only separate trst_push_pull
do_board_reset_init
Info : FlashPro ports available: usb86709, S200XTYRZ3
Info : FlashPro port used: S200XTYRZ3
```

To use a specific FlashPro device when there is more than one connected In the debug launch configuration change the following:

```
--command "set DEVICE M2S090"
--file board/microsemi-cortex-m3.cfg
```

to this which specifies which FlashPro programmer/port to use for debugging:

```
--command "set DEVICE M2S090"
--file board/microsemi-cortex-m3.cfg
--command "microsemi_flashpro port usb86709"
```

Note: The `microsemi_flashpro_port` command must appear after the board script has been specified because this script sources the `interface/microsemi-flashpro.cfg` script.

Debugging using a non FlashPro JTAG interface

By default the Microsemi OpenOCD board scripts (e.g. `board/microsemi-cortex-m3.cfg`) specify that a FlashPro programmer will be used for debugging:

```
# FlashPro
source [find interface/microsemi-flashpro.cfg]

# Device
source [find target/microsemi-cortex-m3.cfg]

# Board specific initialization
proc do_board_reset_init {} {
}
```

This is akin to assuming that all boards come with an on-board FlashPro programmer even if some use a discrete/external programmer. This is the normal and recommended debugging setup.

In this case the debug launch configuration will look something like this:

```
--command "set DEVICE M2S090"
--file board/microsemi-cortex-m3.cfg
```

However it is possible to use any other RVI compatible JTAG probe that OpenOCD supports by default. As an example, to debug using the Olimex ARM-USB-TINY-H

1. In the debug launch configuration put the following:

   ```
   --command "set DEVCE M2S090; set FPGA_TAP N; set FLASHPRO N"
   ```

```
--file board/microsemi-cortex-m3.cfg
--file interface/ftdi/olimex-arm-usb-tiny-h.cfg
```

2. Ensure that the board's JTAG_SEL signal is tied low for RVI (for RVI debugging) rather than high (for FlashPro debugging via the system controller).

3. Connect the Olimex ARM-USB-TINY-H programmer to the board's RVI connector and the USB end to the computer. Ensure that the required drivers are installed. Debugging can now be done via the Olimex ARM-USB-TINY-H device.

The same approach can be taken with other JTAG programmers supported by OpenOCD.

How to connect to/debug a running program

In some situations it is desirable to connect to a program already running on the target without resetting the target, loading the program, executing from the startup code, breakpointing at `main()` etc. To enable this form of debugging:

1. The program/project built must match the program running on the target – i.e. exactly the same code, linker script etc.

2. On the *Startup* page of the debug launch configuration...

3. Clear the *Initial Reset* checkbox

4. In the *Initialization Commands* text field enter `monitor halt`

5. Clear the *Load Symbols and Executable* > *Load Executable* checkbox

With these settings when the debug session is launched SoftConsole the program remains running and the *Suspend* “pause” button can be used to halt it and thereafter normal debugging operations can be performed.

Troubleshooting

If the debug session fails to run as expected then check the following:

a. On Linux was the udev rules file installed to grant non root access to users in the relevant group (usually `plugdev`)?
b. Is a FlashPro device connected (FlashPro 5 on Linux, FlashPro3/4/5 on Windows)?
c. Is there more than one FlashPro device connected? If so SoftConsole may not be using the correct one. If you want to use a specific one of a number of FlashPro devices connected then you can add `--command "microsemi_flashpro port <fp-port-name>"` to the OpenOCD command line options.
d. On Windows did a previous FlashPro3/4 debug session fail leaving OpenOCD (`openocd.exe`) running because `abiactel.dll` did not exit cleanly thus blocking access to the FlashPro device? Check Task Manager/ProcessExplorer for `openocd.exe` and if it’s still running then unplug the FlashPro USB cable and then reattach it and OpenOCD should terminate.
e. If the debug session starts but the program does not run/behave as expected then check that the project was updated to match the target hardware by having the Libero SoC generated firmware and `drivers_config` copied in before rebuilding.
f. Ensure that the relevant CMSIS/HAL firmware core is used.

Other Features

Cortex-M semihosting

Semihosting allows I/O (e.g. file I/O, standard I/O etc.) operations on the target board to be redirected to the SoftConsole host via OpenOCD and the debugger. For example this allows stdio input and output to be performed via the SoftConsole GDB console and allows the program running on the target to read/write files on the host filesystem.

The I/O operations on the target are trapped by library code running on the target and redirected to the host. In order to use semihosting a number of steps must be taken:

- Under *Project* > *Properties* > *C/C++ Build* > *Settings* > *Tool Settings* > *Cross ARM C/C++ Linker* > *Miscellaneous* > *Other linker flags* add `--specs=rdimon.specs` in order to link the libraries required for semihosting.
- The file `CMSIS/startup_gcc/newlib_stubs.c` clashes with the semihosting library support so must be deleted from the project or excluded from the build (check *Properties* > *C/C++ Build* > *Exclude resource from build*) otherwise the program will not link.
- The following code must be added (e.g. to `main.c`):

```
#include <stdio.h>

extern void  initialise_monitor_handles(void);

int main()
{
    ...
    initialise_monitor_handles();
    ...
    iprintf("Hello, World\n");
    ...
}
```

- Programs that use semihosting must be run under the debugger and will not run standalone with no debugger attached as they will hang in the library code that traps I/O operations and attempts to redirect them to the host debugger.
- By default semihosting output is buffered until a `'\n'` is output. This can be overridden to force character granularity output using `setvbuf(stdout, NULL, _IONBF, 0);` but the output will be much slower due to the overhead of many additional semihosting trap operations.

Integer only newlib support

SoftConsole bundles newlib standard library support (https://sourceware.org/newlib/).

It is often possible to build embedded programs in constrained resource (CPU, memory etc.) environments without linking in any standard library overhead. However where standard library support must be used newlib offers a couple of ways to reduce the overhead:

- Smaller integer only `*iprintf()` APIs (e.g. iprintf(), siprintf(), fiprintf() etc.) that avoid the significant additional overhead of floating point support. Refer to the newlib documentation for more information.
- Nano newlib which is a cut down version of the standard newlib library. To use newlib-nano go to the project properties and check the *C/C++ Build* > *Settings* > *Tool Settings* > *Cross ARM C/C++ Linker* > *Miscellaneous* > *Use newlib-nano (--specs=nano.specs)* option.

Static stack profiling

GCC supports static stack usage analysis/profiling.

See here for more on this and add the relevant options to the project settings as required:

https://gcc.gnu.org/onlinedocs/gnat_ugn_unw/Static-Stack-Usage-Analysis.html

Known Issues

Know issues documented in this section are under active investigation to ascertain the root cause and to resolve the underlying problems with the intention that these are resolved in a future release.

Debug launch configuration settings differ for Cortex-M and RISC-V

Be aware that the debug launch configuration settings are different for Cortex-M and RISC-V targets as explained above. The default settings may not automatically match the target CPU. Care must be taken to ensure that the correct configuration settings are applied especially on the Debugger tab. The easiest way to avoid problems is to use the example workspace debug launch configurations as a guide or copy the appropriate one and then customise and specific settings.

Windows occasionally crashes when plugging FlashPro in/out

It has been observed in some cases that plugging a FlashPro JTAG programmer in/out of a Windows machine can sporadically/occasionally cause it to “Blue Screen” (“Blue Screen of Death” or “BSOD”). When this happens the error is often a PAGE_FAULT_IN_NONPAGED_AREA in ftdibus.sys but in some cases a different cause may be displayed.

OpenOCD crashes when attempting to debug RISC-V

In some cases OpenOCD may crash when attempting to debug a RISC-V target. This happens when the debug session would fail anyway due to everything not being order for it to work – for example, the target board is not connected or powered up or the wrong target board is connected. In some cases such a crash may necessitate closing SoftConsole and restarting it in order for a subsequent debug session to work.

RISC-V C++ support

At the moment only C (including mixed C and assembly language) projects will work for RISC-. C++ projects will not work within the SoftConsole Eclipse/CDT environment. The underlying RISC-V GNU toolchain does support C++ but the RISC-V Eclipse Plugin for RISC-V GNU Toolchain does not yet properly support C++ projects.

Invalid command name "arm" when debugging RISC-V

If the debug launch configuration option *Startup* > *Initialization Commands* > *Enable ARM semihosting* is checked/enabled when debugging a RISC-V target then the following error will be displayed by OpenOCD but this can be safely ignored or the *Enable ARM semihosting* option simply unchecked/disabled:

```
invalid command name "arm"
```

Initial startup may be slow

SoftConsole may be slow to start up when run for the first time after installation. The splash screen may be displayed for a period of time before the GUI proper appears. Please be patient if this happens. It is a once off issue that does not happen on subsequent launches.

Flash Programming

OpenOCD has been enhanced to add support for program download to and debugging from SmartFusion eNVM, SmartFusion2 eNVM and Fusion eNVM.

OpenOCD supports programming CFI (Common Flash Interface) external flash parts but not non CFI external flash.

No unlocking or locking of eNVM pages is carried out when downloading to eNVM. eNVM pages to be modified are expected and assumed to be unlocked.

Build Project context menu option sometimes disabled

Sometimes the *Build Project* option in the context menu that appears when right clicking on a project is disabled when it should be enabled. This seems to be a CDT bug. If this happens right click on another node in the *Project Explorer* tree view and then back onto the project in question and it will be re-enabled. Alternatively use the *Build* toolbar (hammer) icon to select and build a specific project build target.

Windows firewall and OpenOCD

On Windows if there is a firewall in use then the first time that a debug session is run the firewall may prompt that it is blocking OpenOCD. Allow the firewall to unblock it and save this as the default setting if necessary.

Multiple debug sessions

Only one debug session should be active at any one time. If a deliberate or inadvertent attempt is made to run more than one then SoftConsole may not work properly and it may be necessary to exit and restart SoftConsole for further debugging to work properly.

Memory Monitor fails to display

There have been unconfirmed reports that in some cases an attempt to configure/enable a Memory Monitor will fail and the debug session may not operate correctly subsequently. If this happens then exit and restart SoftConsole.

Warning when installing Windows FlashPro drivers

On some Windows installations the FlashPro drivers installer (FPdrivers.exe) launched from the SoftConsole installer may display a warning “Windows can’t verify the publisher of this driver software”. If this happens please select the “Install this driver anyway” option.

FlashPro JTAG debugging is unreliable on virtual machines

FlashPro JTAG debugging is unreliable on virtual machines so it is recommended that only physical machines and not virtual machines be used for SoftConsole debugging.

Invalid Project Path warnings in Cortex-M projects

Occasionally and sporadically Eclipse may display “Invalid Project Path” warnings in Cortex-M projects for no obvious reason. For example:

```
Invalid project path: Duplicate path entries found (/fpga-cortex-m1-blinky [Include path]
base-path:fpga-cortex-m1-blinky isSystemInclude:true)
Invalid project path: Include path not found (smartfusion2-cortex-m3-blinky\#undef
__ARM_FEATURE_CRYPTO)
```

Deleting these warnings may eliminate them. But if they continue to appear then, for the moment, just ignore them.

“DAP transaction stalled (WAIT)” messages when debugging SmartFusion2 Cortex-M3

When debugging a SmartFusion2 Cortex-M3 target where the SmartFusion2 envm boot area does not contain a valid Cortex-M3 program (for example zeroized or garbage envm contents) then one or more instances of the following message may appear in the OpenOCD log:

```
Info : DAP transaction stalled (WAIT) - slowing down
```

This arises because if the Cortex-M3 boots from zeroized or garbage envm it can end up in a double fault/lockup/reset cycle and the debugger may experience delays while trying to reset it. However the debugger will reset the target and these messages can be safely ignored.

Error: Got exception when reading some RISC-V registers

Not all RISC-V registers are implemented in all RISC-V targets. For example RISC-V targets with no hardware floating point support (no F, D or Q extension support) do not implement any FPU (Floating Point Unit) registers. Similarly not all Control/Status Registers (CSRs) are implemented in all cases. When an attempt is made to read a register that does not exist then OpenOCD will display a message of the form:

```
Error: Got exception 0xffffffff when reading register ...
```

Such error messages can be safely ignored.

Debugging and multiple device JTAG chains

Debugging is not yet possible in a multiple device JTAG chain where one device (a Microsemi FPGA) in the chain contains a Cortex-M1 or RISC-V CPU to be targeted. Where the JTAG chain comprises a single Microsemi FPGA device containing multiple instances of a soft CPU core then a specific CPU instance can be targeted for debugging in that case.

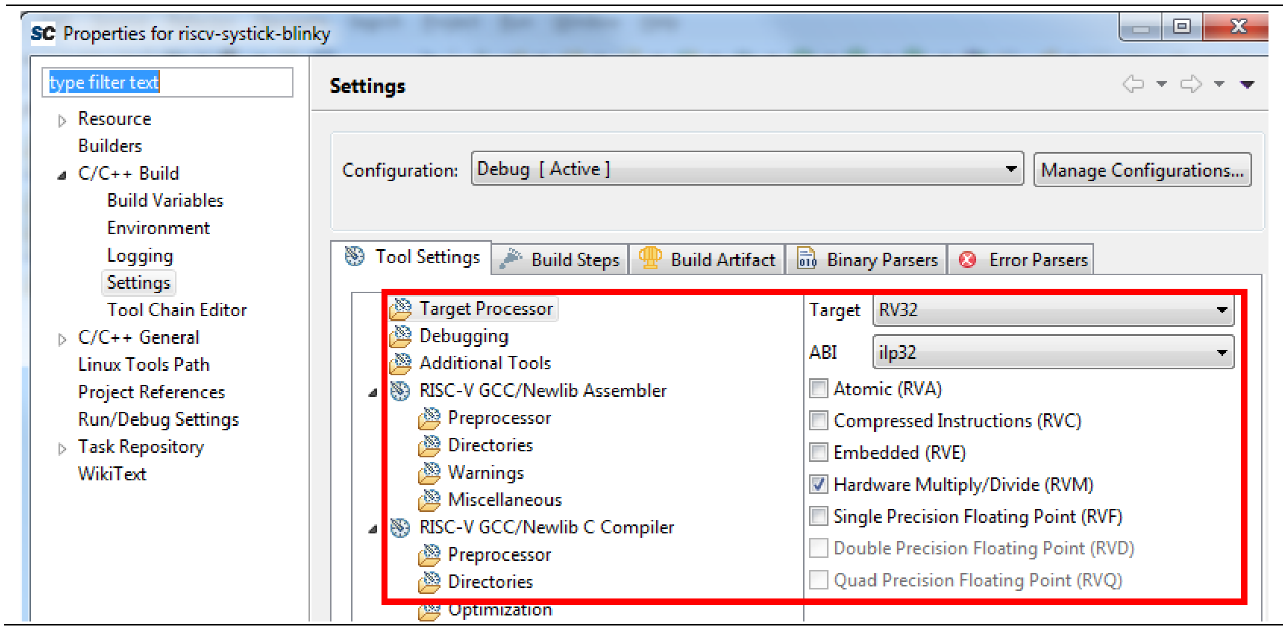
Debugging a particular SmartFusion or SmartFusion2 Cortex-M3 in a multiple device chain can be achieved through judicious and appropriate customization of the OpenOCD board script to include a description of other device TAPs in the JTAG chain.

For example make a copy of the `<SoftConsole-install-dir>/openocd/share/openocd/scripts/board/microsemi-riscv.cfg` board script and modify it to declare any device TAPS before and/or after the device containing the CPU which is to be debugged. The following example describes a three M2S090 device chain where the middle device contains the Cortex-M3 to be debugged:

```
# FlashPro
source [find interface/microsemi-flashpro.cfg]

# Ignore leading device
jtag newtap M2S090_0 tap -irlen 8 -expected-id 0x0f8071cf -ignore-version

# Want to debug the Cortex-M3 in this device
source [find target/microsemi-cortex-m3.cfg]

# Ignore trailing device
jtag newtap M2S090_2 tap -irlen 8 -expected-id 0x0f8071cf -ignore-version

# Board specific initialization
proc do_board_reset_init {} {
}
```

RISC-V target support

The primary RISC-V target is the Microsemi RV32IM CPU. However the RISC-V tools bundled with SoftConsole (in particular the Eclipse Plugin for RISC-V GNU Toolchain and the RISC-V GNU Toolchain itself) should be capable of supporting many other RISC-V RV32 and RV64 targets. However not all RISC-V targets have been tested in practice so if issues arise please report them to Microsemi technical support or via the RISC-V mailing lists (https://riscv.org/mailing-lists/). The Eclipse Plugin for RISC-V GNU Toolchain offers various configuration options in the project properties that control the target CPU, bit size, ABI and extensions.



Configuration: Debug [Active]

Target: RV32
ABI: ilp32
[ ] Atomic (RVA)
[ ] Compressed Instructions (RVC)
[ ] Embedded (RVE)
[✓] Hardware Multiply/Divide (RVM)
[ ] Single Precision Floating Point (RVF)
[ ] Double Precision Floating Point (RVD)
[ ] Quad Precision Floating Point (RVQ)

SoftConsole v3.4 or earlier workspaces/projects

SoftConsole v3.4 or earlier workspaces, projects and debug launch configurations are not compatible with this version of SoftConsole and must be recreated.

SoftConsole v5.0 RISC-V projects and debug launch configurations

Due to changes to the Eclipse Plugin for RISC-V GNU Toolchain since SoftConsole v5.0 was released it is possible that RISC-V projects created using SoftConsole v5.0 may not work correctly in SoftConsole v5.1. For this reason it is recommended that existing projects created in SoftConsole v5.0 or any pre-release version of SoftConsole v5.x are recreated in SoftConsole v5.1. Note that SoftConsole v5.1 RISC-V debug launch configurations require `-f board/microsemi-riscv.cfg` whereas some pre-release versions of SoftConsole v5.x used a different board script name (for example `-f board/microsemi-riscv-rv32im.cfg`). If there are any problems using existing RISC-V debug launch configurations then recreate them using one of the example workspace RISC-V debug launch configurations as a guide.

Other useful Documentation

1. Erich Styger's "MCU on Eclipse" blog (http://mcuoneclipse.com/): Useful tips and tricks for using Eclipse/CDT, GNU ARM Eclipse, GNU Tools for ARM Embedded Processors, OpenOCD etc. The Compendium page is a good place to find posts/articles relevant to Eclipse, OpenOCD etc.
2. The websites and documentation links for the various open source components used in SoftConsole are also useful references. These are listed below.

Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call **800.262.1060**
From the rest of the world, call **650.318.4460**
Fax, from anywhere in the world **408.643.6913**

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

For Microsemi SoC Products Support, visit http://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support.

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group home page, at http://www.microsemi.com/soc/.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office. Visit About Us for sales office listings and corporate contacts.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense & security, aerospace and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif., and has approximately 3,600 employees globally. Learn more at **www.microsemi.com**.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA

Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

E-mail: sales.support@microsemi.com

XXXXXXXX-06Jun2017