# Microsemi SoftConsole v6.0

## Release Notes

# Table of Contents

# Microsemi SoftConsole v6.0

## Introduction

These are release notes for Microsemi SoftConsole v6.0.

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_v6.0` on Windows or `$HOME/Microsemi_SoftConsole_v6.0` on Linux).

## Overview

### Key features

- Runs on Windows and Linux.
- Development/debug support for Microsemi PolarFire SoC RISC-V 64-bit multi-processor, Microsemi Mi-V RISC-V and Arm® Cortex®-M CPUs/SoCs.
- 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 Microsemi PolarFire SoC, Microsemi SmartFusion® and SmartFusion2 Arm® Cortex-M3, Microsemi Arm® Cortex-M1 and Microsemi Mi-V RISC-V firmware development and debugging.
- Uses OpenOCD for Arm Cortex-M 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 (FlashPro3/4/5 on Windows, FlashPro5 on Linux).
- Supports Arm Cortex-M semi-hosting redirection of standard/file I/O from target board to host debugger.
- Includes Antmicro's Renode emulation platform which can be used to emulate PolarFire SoC and Mi-V projects.
- Includes a built-in terminal emulator for connecting to a target board's serial port.
- Includes cppcheck and companion cppcheclipse plugin integration for static code analysis.
- Supports newlib nano for Cortex-M and RISC-V providing an even more lightweight standard library implementation ideal for resource constrained environments.
- Suitable for development and debug of bare metal and lightweight RTOS based embedded applications.

### 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.
- Compatibility with any 32bit Operating Systems, SoftConsole v6.0 needs a 64bit Operating System.
- Compatibility with SoftConsole v5.1 RISC-V projects. Due to a change in the RISC-V Eclipse plugin support between SoftConsole v5.1 and SoftConsole v5.2, SoftConsole v5.1 RISC-V projects will not work in SoftConsole v5.2. Such projects must be recreated using the same project source files and settings to work in SoftConsole v5.2.
- Debugging a CoreJTAGDebug/UJTAG Cortex-M1 or Mi-V RISC-V CPU that is in one device in a multiple device JTAG chain.
- Debugger driven download of programs to non-CFI external parallel flash memories.
- Core8051/Core8051s firmware development and debugging. Use Keil C51 Development Tools with Core8051/Core8051s ISD-51 support.

## Quick start guide

1. Read these release notes in full.
2. Download the SoftConsole installer from the Microsemi website. Verify the downloaded file integrity using the checksum file provided.
3. Follow the installation instructions below for the relevant OS platform. If installing on Linux pay careful attention to the pre-install and post-install instructions.
4. Run SoftConsole from the desktop shortcut or “Start” menu entries created by the installer. This will launch SoftConsole and open the example workspace.
5. Do not use older or previous workspaces. Use the bundled workspaces, either workspace.examples which has bundled examples or workspace.empty which has no imported projects .
6. 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.
7. 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.
8. Use the Microsemi website (https://www.microsemi.com/product-directory/product-support/4217-fpgas-socs-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 Libero and SoftConsole projects.

# Supported platforms

- Operating systems - 64bit only

  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 (deprecated, support will be removed in the future releases)
      - 7
    - Ubuntu
      - 14.04 LTS
      - 16.04 LTS
    - openSUSE
      - LEAP 15
    - Debian
      - 9

- CPUs
  - Microsemi PolarFire SoC RISC-V 64-bit multi-processor FPGA
  - Microsemi Mi-V RISC-V CPU soft cores for PolarFire, RTG4, IGLOO2 and SmartFusion2 FPGAs
  - Microsemi SmartFusion2 Arm Cortex-M3
  - Microsemi SmartFusion Arm Cortex-M3
  - Microsemi Arm Cortex-M1 for RTG4 and PolarFire FPGAs
  - Microsemi Arm Cortex-M1 for M1 IGLOO, ProASIC3, ProASIC3L and Fusion FPGAs

- Boards
  - Microsemi PolarFire SoC RISC-V 64-bit multi-processor FPGA
    - Microsemi HiFive Unleashed Platform
    - Renode Microsemi PolarFire SoC emulation model
  - Mi-V RISC-V CPU soft cores
    - IGLOO2 RISC-V Creative Development Board (M2GL025)
    - IGLOO2 Evaluation Kit (M2GL010)
    - SmartFusion2 Security Evaluation Kit (M2S090)
    - SmartFusion2 Advanced Development Kit (M2S150)
    - RTG4 Development Kit (RT4G150)
    - PolarFire Evaluation Kit (MPF300TS)
  - SmartFusion2 Arm Cortex-M3
    - SmartFusion2 FPGA Development Kits and Boards
  - SmartFusion Arm Cortex-M3
    - SmartFusion FPGA Kits
  - Arm Cortex-M1 for RTG4 and PolarFire FPGAs
    - RTG4 Development Kit (RT4G150)
    - PolarFire Evaluation Kit (MPF300TS)
  - Arm Cortex-M1 for M1 FPGAs
    - Fusion Embedded Development Kit (M1AFS1500)

- JTAG Debug
  - Microsemi FlashPro3, FlashPro4 and FlashPro5 on Windows

  - Microsemi FlashPro5 on Linux
  - Olimex ARM-USB-TINY-H
  - 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 v11.9 SP1
    - Microsemi Libero SoC PolarFire v2.3
    - Microsemi Firmware Catalog v11.9 SP1
  - Firmware (minimum required version)
    - PolarFire SoC PSE_HAL 1.0.10
    - RISC-V Hardware Abstraction Layer (HAL) 2.2.103
    - SmartFusion2 CMSIS Hardware Abstraction Layer 2.3.105
    - SmartFusion CMSIS-PAL 2.4.102
    - Cortex-M1 CMSIS Hardware Abstraction Layer 2.0.101
    - (DirectCore) Hardware Abstraction Layer 2.3.102

# Free/Open source packages

## Packages used

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

| Oracle Java SE | |
|---|---|
| Version | 10.0.2 |
| 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<br>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.<br>Credit/thanks to Oracle. |
| **Eclipse/CDT** | |
| Version | Eclipse 4.8 (Photon) + CDT 9.5.2 for Eclipse Photon |
| Home page | https://www.eclipse.org/downloads/packages/release/photon/r |
| Documentation | https://help.eclipse.org/photon/index.jsp |
| License | Eclipse Public License v2.0<br>https://www.eclipse.org/legal/epl-2.0/ |
| Notes | Eclipse/CDT, in conjunction with the GNU MCU Eclipse plugins, provide the main SoftConsole GUI Integrated Development Environment.<br>The Windows Eclipse/CDT starter.exe has been modified by Microsemi to allow for graceful termination of OpenOCD launched from Eclipse.<br>Credit/thanks to the Eclipse/CDT developer community. |
| **GNU MCU Eclipse Plugins** | |
| Version | v4.4.2-201809300659 |
| Home page | https://gnu-mcu-eclipse.github.io/ |
| Documentation | https://gnu-mcu-eclipse.github.io/ |
| License | Eclipse Public License v1.0<br>https://gnu-mcu-eclipse.github.io/licenses/plug-ins/#eclipse-public-license<br>The copyright owner for all the GNU MCU Eclipse plug-ins is Liviu Ionescu and all rights are reserved. |
| Notes | Formerly GNU Arm Eclipse but renamed to GNU MCU Eclipse ever since the plugins began supporting both Arm and RISC-V CPUs. The following GNU MCU Eclipse plugins are used:<br>• GNU MCU C/C++ Arm 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.<br>• GNU MCU C/C++ RISC-V Cross Compiler: provides specific support for RISC-V targets by way of custom project properties pages and integration with the back-end RISC-V GNU toolchain. |

| | |
|---|---|
| | • GNU MCU C/C++ OpenOCD Debugging: provides specific support for debugging Arm and RISC-V targets using OpenOCD from within the Eclipse environment.<br>Credit/thanks to Liviu Ionescu. |
| **GNU Arm Embedded Toolchain** | |
| Version | 7.3.1-1.1 20180724-0637 (Arm/embdded-7-branch revision 255204) built using the GNU MCU Eclipse Docker based build scripts (https://github.com/gnu-mcu-eclipse/arm-none-eabi-gcc-build) |
| 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 a full GCC based toolchain (including GCC, GDB, binutils, newlib, newlib nano etc.) for Arm Cortex-M and Cortex-R targets.<br>Details of the specific versions of the individual tools in each release package can be found on the Arm Developer website.<br>Credit/thanks to Arm, the GNU Arm Embedded Toolchain development community and Liviu Ionescu. |
| **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.<br>Credit/thanks to Arm. |
| **RISC-V GNU Toolchain** | |
| Version | GCC 7.2.0 based RISC-V github (https://github.com/riscv/riscv-gnu-toolchain) built using the GNU MCU Eclipse Docker based build scripts (https://gnu-mcu-eclipse.github.io/toolchain/riscv/build-procedure/). |
| 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 bare metal targets and the following multilibs:<br>• march=rv32i/mabi=ilp32<br>• march=rv32ia/mabi=ilp32<br>• march=rv32iac/mabi=ilp32<br>• march=rv32iaf/mabi=ilp32f<br>• march=rv32iafc/mabi=ilp32f<br>• march=rv32ic/mabi=ilp32<br>• march=rv32if/mabi=ilp32f<br>• march=rv32ifc/mabi=ilp32f |

| | |
|---|---|
| | • march=rv32im/mabi=ilp32<br>• march=rv32ima/mabi=ilp32<br>• march=rv32imac/mabi=ilp32<br>• march=rv32imaf/mabi=ilp32f<br>• march=rv32imafc/mabi=ilp32f<br>• march=rv32imc/mabi=ilp32<br>• march=rv32imf/mabi=ilp32f<br>• march=rv32imfc/mabi=ilp32f<br>• march=rv64i/mabi=lp64<br>• march=rv64ia/mabi=lp64<br>• march=rv64iac/mabi=lp64<br>• march=rv64iaf/mabi=lp64f<br>• march=rv64iafc/mabi=lp64f<br>• march=rv64iafd/mabi=lp64d<br>• march=rv64iafdc/mabi=lp64d<br>• march=rv64ic/mabi=lp64<br>• march=rv64if/mabi=lp64f<br>• march=rv64ifc/mabi=lp64f<br>• march=rv64ifd/mabi=lp64d<br>• march=rv64ifdc/mabi=lp64d<br>• march=rv64im/mabi=lp64<br>• march=rv64ima/mabi=lp64<br>• march=rv64imac/mabi=lp64<br>• march=rv64imaf/mabi=lp64f<br>• march=rv64imafc/mabi=lp64f<br>• march=rv64imafd/mabi=lp64d<br>• march=rv64imc/mabi=lp64<br>• march=rv64imf/mabi=lp64f<br>• march=rv64imfc/mabi=lp64f<br>• march=rv64imfd/mabi=lp64d<br>• march=rv64imfdc/mabi=lp64d<br><br>Note that this suite of multilibs supports the “natural” ABI for each architecture.<br><br>The “native” mode of the tools is RV64GC in the absence of any -march/-mabi flags being passed to the tools.<br><br>Credit/thanks to the RISC-V, GCC, LD, GDB, binutils, newlib, and GNU MCU Eclipse development communities especially Liviu Ionescu. |
| **OpenOCD** | |
| Version | v0.10.0+dev-00352-gaa6c7e9b-dirty (2018-10-22-10:08) + mods (see notes) |
| Home page | http://openocd.org/ |
| Documentation | http://openocd.org/documentation/ |
| License | GNU General Public License v2<br>https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html |
| Notes | OpenOCD sits between GDB and the target hardware (JTAG debug probe, target board and CPU) to allow for program download and debug on real hardware rather than a simulated or emulated target. 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 (RSP) 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.<br>The base OpenOCD v0.10.0 is supplemented by modifications by<br>• Microsemi – to add support for FlashPro, SmartFusion2/SmartFusion/Fusion envm, finding scripts relative to OpenOCD bin directory, other fixes and enhancements<br>• SiFive (https://www.sifive.com/) – to add support for RISC-V debugging via RISC-V External Debug Support v0.11 and v0.13<br>• GNU MCU Eclipse – other modifications and Docker based build scripts<br>Credit/thanks to the OpenOCD development community and to SiFive. |
| **GNU MCU Eclipse Build Tools (Windows only)** | |
| Version | v2.10-20180103 |
| Home page | https://gnu-mcu-eclipse.github.io/windows-build-tools/ |
| Documentation | https://gnu-mcu-eclipse.github.io/windows-build-tools/ |
| License | https://gnu-mcu-eclipse.github.io/licenses/tools/<br>make:<br>GNU General Public License v3<br>http://www.gnu.org/copyleft/gpl.html<br>BusyBox<br>GNU General Public License v2<br>https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html |
| Notes | Provides common Unix style utilities such as cp, echo, make etc. for Windows.<br>Credit/thanks to Liviu Ionescu and the original developers of make and BusyBox. |
| **Antmicro Renode** | |
| Version | v1.6.0 |
| Home page | https://github.com/renode<br>https://renode.io/ |
| Documentation | https://renode.readthedocs.io/en/latest/ |
| License | MIT License:<br>https://github.com/renode/renode/blob/master/LICENSE |
| Notes | Renode can simulate physical hardware systems - including both the CPU, peripherals, sensors, environment and wireless medium between nodes.<br><br>Credit/thanks to the Antmicro development team. |
| **Cppcheck** | |
| Version | v1.83 |
| Home page | http://cppcheck.sourceforge.net/ |
| Documentation | https://sourceforge.net/p/cppcheck/wiki/Home/ |
| License | GNU Public License v3<br>http://www.gnu.org/copyleft/gpl.html |
| Notes | Cppcheck is a static analysis tool for C/C++ code. It detects the types of bugs that the compilers normally fail to detect. The goal is no false positives.<br>Credit/thanks to Daniel Marjamäki. |

| Cppcheclipse | |
|---|---|
| Version | v1.1.0 |
| Home page | https://github.com/kwin/cppcheclipse |
| Documentation | https://github.com/kwin/cppcheclipse/wiki |
| License | Apache 2.0 https://github.com/kwin/cppcheclipse/blob/master/LICENSE |
| Notes | Cppcheclipse is an Eclipse plugin which integrates cppcheck with a CDT project. Cppcheclipse can run/configure cppcheck from the Eclipse UI. |
| **Inno Setup (Windows only)** | |
| Version | Inno Setup QuickStart Pack v5.6.1-unicode |
| Home page | http://www.jrsoftware.org/isdl.php |
| Documentation | http://www.jrsoftware.org/ishelp/ |
| License | Inno Setup License<br>http://www.jrsoftware.org/files/is/license.txt |
| Notes | Inno Setup is used to create the SoftConsole installer for Windows.<br>Credit/thanks to Jordan Russell. |
| **InstallJammer (Linux only)** | |
| Version | v1.2.15 |
| Home page | https://en.wikipedia.org/wiki/InstallJammer |
| Documentation | http://installjammer.com/docs/ (domain expired)<br>https://sourceforge.net/p/installjammer/wiki/Home/ |
| License | GNU General Public License v2 with exception |
| Notes | InstallJammer is used to create the SoftConsole installer for Linux.<br>Credit/thanks to the InstallJammer development community. |

# Installation

## Windows

### Installing

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

The installer is a 64-bit executable GUI based program named `Microsemi-SoftConsole-v6.0.0.x-Windows-Installer.exe` (where x is the build number). It must be run with admin privileges. Run the installer and follow the GUI installer wizard instructions on screen.

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

There may be a slight pause completing the SoftConsole installation after the FlashPro drivers installer has completed – this is normal.

The Renode emulation platform can be used out of the box on Windows 8.1 and Windows 10, but on Windows 7 Microsoft ,NET v4.7.2 must be installed first:

https://www.microsoft.com/net/download/dotnet-framework-runtime

## 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`.

### Installing

SoftConsole is a 64-bit application with one 32-bit component, therefore it requires a 64bit operating system and can be installed without any prerequisite steps.

1. The installer is a 64-bit executable GUI based program named `Microsemi-SoftConsole-v6.0.0.x-Linux-x86_64-Installer` (where x is the build number).

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-v6.0.0.x-Linux-x86_64-Installer
   ```

3. Run the installer:

   ```
   ./Microsemi-SoftConsole-v6.0.0.x-Linux-x86_64-Installer
   ```

4. If errors of the following form appear:

   ```
   can't invoke "winfo" command:  application has been destroyed
       while executing
   "winfo exists $info(Wizard)"
   ...
   ```

   then run the installer in command line mode as follows:

   ```
   ./Microsemi-SoftConsole-v6.0.0.x-Linux-x86_64-Installer --mode console
   ```

5. Follow the installer GUI wizard or console mode instructions on screen. If necessary, run the installer in debug mode to diagnose problems with running it:

```
./Microsemi-SoftConsole-v6.0.0.x-Linux-x86_64-Installer --debugconsole
```

6. If, after installing, the desktop shortcuts do not appear or work correctly then log out and back in again first.
7. If still there is a problem, make sure the `xdg-utils` package is installed and reinstall the SoftConsole. The installer requires the `xdg-utils` package to create shortcuts and menus, but the supported distributions have it already installed by default.

## After installing

To use debugging with hardware targets a single 32-bit package needs to be installed. The following commands must be run with root/sudo provileges:

Ubuntu/Debian:

```
dpkg --add-architecture i386 && apt-get update && apt-get install
libstdc++6:i386
```

CentOS/Red Hat Enterprise Linux:

```
yum update && yum install libstdc++.i686
```

openSUSE:

```
zypper install libstdc++6-32bit
```

Notes:

1. Many platforms have GCC and `make` packages installed by default, but it is advisable to make sure that these are installed.

   Ubuntu/Debian:

   ```
   apt-get install build-essential
   ```

   CentOS/Red Hat Enterprise Linux:

   ```
   yum groupinstall 'Development Tools'
   ```

   openSUSE:

   ```
   zypper install -t pattern devel_basis
   ```

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

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.

Further packages are needed if the Renode emulation platform will be used:

1. Visit http://www.mono-project.com/download/stable/ and follow the instructions for your repository.
2. Install `mono-complete` instead of `mono-develop`
3. Some distributions need GTK related packages.

Debian/Ubuntu:

```
apt-get install mono-complete gtk-sharp2 libcanberra-gtk-module
```

CentOS/Red Hat Enterprise Linux:

```
yum install mono-complete.x86_x64 gtk-sharp2.x86_64 libcanberra-gtk2.x86_64
```

CentOS 6 contains an old version of the `cairo` 2D graphics library package and without upgrading to 1.9.4 or newer frequent UI crashes can happen as it is responsible for 2D drawing routines. Downloading the later RPM and installing it manually from here can solve this problem:
http://rpm.pbone.net/index.php3/stat/4/idpl/33561465/dir/centos_6/com/cairo-1.10.0-9.21.x86_64.rpm.html

CentOS 7 unstable UI can be solved by switching to GTK3.0 - see Glitches, unstable UI and cosmetic issues on Linux section.

openSUSE:

```
zypper install mono-complete gtk-sharp2 libcanberra-gtk2-module
```

openSUSE Leap 42.3 and 15 are not supported by the Mono project and the packages supplied with the distribution must be used. Leap 15 includes Mono v5 which is what Renode requires. But Leap 42.3 contains the older v4.6.x with which Renode may experience warning messages, UI issues and reduced usability of the emulator. Ubuntu 17.10 is not supported by the Mono project and has to resort to the default repository, which includes the older Mono v4.6,x and might experience issues as well.

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

1. 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 non-root users:

   ```
   cd <SoftConsole-install-dir>/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. Some distributions do not create the plugdev group and add users to it automatically. In that case the plugdev group must be created manually and the user added to the plugdev group. Refer to this link for instructions on how to do this:
   https://www.howtogeek.com/50787/add-a-user-to-a-group-or-second-group-on-linux/

2. 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.

3. To check that FlashPro5 can be used without root privileges…

   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 <SoftConsole-install-dir>/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 similar to the following:

```
Open On-Chip Debugger
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 similar to the following:

```
Open On-Chip Debugger
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 Mi-V RISC-V the output should be similar to the following:

```
Open On-Chip Debugger
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 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>/fpServer/bin
./fpServer

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 v11.9 or later to create hardware designs and to export firmware drivers and example projects.

For PolarFire FPGAs use Microsemi Libero SoC PolarFire v2.3 or later.

The Microsemi Firmware Catalog can be used to generate firmware drivers and example projects for use in SoftConsole v6.0.

## 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.

- PolarFire SoC PSE_HAL v1.0.10
- RISC-V Hardware Abstraction Layer (HAL) 2.2.103
- SmartFusion2 CMSIS Hardware Abstraction Layer 2.3.105
- SmartFusion CMSIS-PAL 2.4.102
- Cortex-M1 CMSIS Hardware Abstraction Layer 2.0.101
- (DirectCore) Hardware Abstraction Layer 2.3.102

**Warning:**

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

### 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 for the SoftConsole project to match and be compatible with the target hardware.

The most convenient way to avoid mismatch 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.

In some cases, the firmware project will define target specific details such as clock speeds, UART baud divisors etc. that must match the target hardware for proper functionality.

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`, `pse_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 or later and should not be used.

- SoftConsole v5.1 RISC-V projects are not compatible with SoftConsole v5.2 or later and must be recreated using the same source files and equivalent project settings for use in SoftConsole v5.2 or later.

## 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 non-root users.

## SoftConsole v3.4

SoftConsole v3.4 workspaces, projects and debug launch configurations are not compatible with SoftConsole v5.1 or later 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 v6.0. However, this is not an onerous task and is explained elsewhere in the release notes.

Similarly, SoftConsole v6.0 workspaces, projects and debug launch configurations are not compatible with SoftConsole v3.4.

## SoftConsole v4.x

SoftConsole v4.x workspaces, projects and debug launches will work in SoftConsole v5.x or later but it is advisable to check all configuration settings to ensure that they are appropriate.

## SoftConsole v5.1 RISC-V projects

SoftConsole v5.1 RISC-V projects and debug launch configurations are not compatible with SoftConsole v5.2 or later and must be recreated. This is because SoftConsole v5.2or later uses different Eclipse plugins for RISC-V support than SoftConsole v5.1.

## Microsemi github

The Microsemi github (https://github.com/RISCV-on-Microsemi-FPGA) is a useful reference for example/demo/reference RISC-V Libero and SoftConsole projects and for other up to date information about Microsemi Mi-V ecosystem resources.

# 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 several 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

- pse-blinky: simple GPIO LED and UART example program targeting the Renode PolarFire SoC emulation platform. The associated tool and debug launch configurations facilitate debugging this example on the emulation platform. Refer to the README for more information.
- 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.
- miv-rv32im-interrupt-blinky: interrupt driven LED blinker and UART echo program for a system containing the Mi-V RISC-V soft processor that supports at least the M extension.
- miv-rv32im-systick-blinky: timer driven LED blinker and UART echo program for a system containing the Mi-V RISC-V soft processor that supports at least the M extension.
- miv-rv32iamf-mandelbrot-uart: Displaying Mandelbrot fractals with ASCII art through UART while using the RISC-V F extension
- miv-rv32iamf-raytracer-uart-cpp: ASCII art raytraced rendering of a sphere while using C++ and RISC-V F extension. Output is displayed over UART, extra attention needs to be paid to the readme and get the UART clocked correctly. The terminal window most likely must be resized to fit the content without any artifacts.
- 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.
- All the Mi-V projects can be run and debugged without target hardware on the Renode emulation platform.
- The projects have enabled cppcheck on each build.

### 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 Mi-V RISC-V targets.

When creating new debug launch configurations for 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.

## Empty workspace

SoftConsole includes an empty workspace which has the new integrated “develop and debug” perspective and identical settings to the example workspace. This workspace is located at:

```
../extras/workspace.empty
```

## Creating a new workspace

To create a new workspace it is recommended to make a copy the `workspace.empty` example workspace and open that. This is because the `workspace.empty` workspace has some useful settings – including an integrated “develop and debug” perspective – preconfigured to make development and debug easier than with a completely new/blank workspace.

# Projects

## Creating a new 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)
3. Select the appropriate toolchain.
   For a Cortex-M project select *Toolchains* = *Arm Cross GCC*.
   For a RISC-V project select *Toolchains* = *RISC-V Cross GCC*.
4. Click *Next* > to go to the next wizard page, *Select Configurations*.

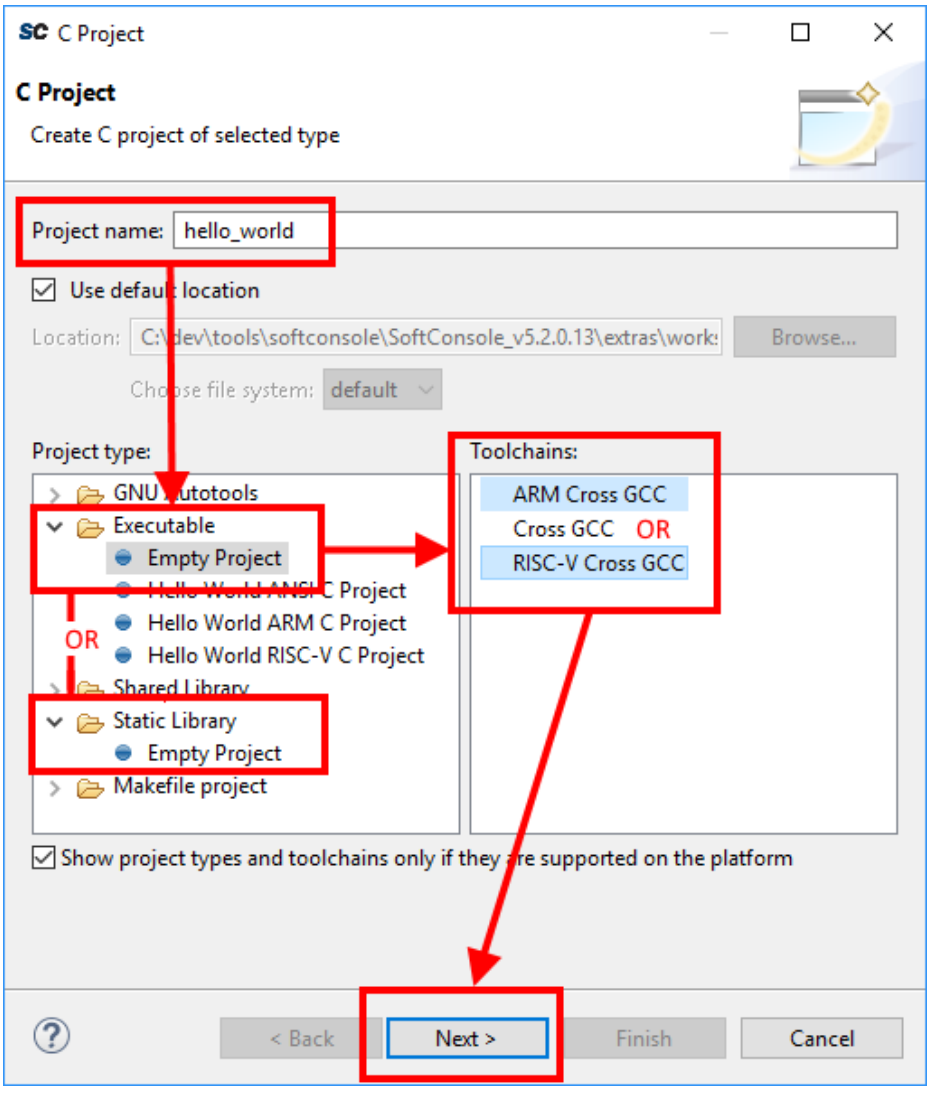



**Figure 1. New Project**

5. The *Select Configurations* page of the wizard allows the configurations or build targets that the project will support to be configured. By default, two configurations are created – *Debug* and *Release*. Should other configurations be required these can be created using the *Advanced settings…* button which launches the project *Properties* dialog in which additional configurations can be specified or properties for any or all configurations can be changed. Normally the default *Debug* and *Release* configurations are sufficient. When finished click *the Next* > button to go to the next wizard page, *GNU Arm Cross Toolchain* or *GNU RISC-V Cross Toolchain*.

6. The *GNU Arm Cross Toolchain* or *GNU RISC-V Cross Toolchain* wizard page specifies the name and path of the toolchain to be used to build the project. These should be correct by default but double check that the values are as follows:

   Cortex-M project:
   *Toolchain name* = `GNU Tools for Arm Embedded Processors (arm-none-eabi-gcc)`
   *Toolchain path* = `${eclipse_home}/../arm-none-eabi-gcc/bin`

   RISC-V project:
   *Toolchain name* = `RISC-V GCC/Newlib (riscv64-unknown-elf-gcc)`
   *Toolchain path* = `${eclipse_home}/../riscv-unknown-elf-gcc/bin`

7. Click *Finish* > to complete the creation of the new project.

## Project Settings

Most of the project settings default to usable values. However, some project settings must be modified manually depending on the target device/CPU. Use the example projects as a guide to creating new project while bearing in mind that these are just simple functional examples and a real application may benefit from the use of some of the many other configuration options and command line options that the underlying GCC tools support.

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 (by default *Debug* and *Release*) or else select a specific configuration (e.g. *Configuration = Debug* or *Configuration = Release*) to configure settings applicable only to that build target.

Except where noted the settings below can be configured for all *[All Configurations]*.

### All CPU targets

**Target Processor**

The characteristics of the target CPU are configured in the project's *Properties* > *C/C++ Build* > *Settings* > *Tool Settings* > *Target Processor* section.

For Cortex-M projects these will default to *Arm Family = cortex-m3* which is correct for SmartFusion2 and SmartFusion2. For Cortex-M1 projects this should be changed to *cortex-m1*.

For RISC-V projects the settings must be configured to match the target CPU characteristics so that the underlying compiler tools are passed the correct `-march=<arch>` and `-mabi=<abi>` options, code is generated in line with the supported and used extensions and the appropriate multilibs are linked. Note that the RISC-V toolchain defaults to targeting rv64gc which may not be appropriate for all RISC-V target designs.

The main options of relevance here are:

*Architecture*: specifies the base architecture – e.g. *RV32** for Mi-V 32-bit soft cores or *RV64** for PolarFire SoC 64-bit multi-processor

*Multiply extension (RVM)*: check if the target supports the M (hardware multiply/divide) extension

*Atomic extension (RVA)*: check if the target supports the A extension

*Floating point*: specifies what hardware floating point extension the target supports

*Compressed extension (RVC)*: check if the target supports the C extension

*Integer ABI*: specifies the integer ABI to be used – usually set to *LP32 (-mabi=ilp32)* for Mi-V or *LP64 (-mabi=lp64)* for PolarFire SoC

*Floating point ABI*: specifies the floating-point ABI to be used

*Code model*: specifies the code model to be used

*Align*: specifies the alignment policy – should be set to *Strict (-mstrict-align)* to avoid unaligned memory access exceptions when using the Microsemi (Mi-V) RISC-V Hardware Abstraction Layer (HAL) or PolarFire SoC PSE_HAL and *-Os* to optimize the program for size

**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.

Cortex-M project:
select *Tool Settings > Cross Arm GNU C/C++ Linker > General*

RISC-V project:
select *Tool Settings > GNU RISC-V Cross 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.:

- PolarFire SoC
  Please refer to the PolarFire SoC PSE_HAL documentation and PolarFire SoC/PSE example project(s) bundled with SoftConsole for guidance on how to configure this option.

- Mi-V RISC-V
  `"${workspace_loc:/${ProjName}/riscv_hal/microsemi-riscv-ram.ld}"`

- SmartFusion2 Cortex-M3:
  `"${workspace_loc:/${ProjName}/CMSIS/startup_gcc/debug-in-microsemi-smartfusion2-esram.ld}"`

- SmartFusion Cortex-M3:
  `"${workspace_loc:/${ProjName}/CMSIS/startup_gcc/debug-in-actel-smartfusion-envm.ld}"`

- Cortex-M1:
  `"${workspace_loc:/${ProjName}/blinky_linker_config.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.
- CMSIS/HAL bundled linker scripts are just examples that can usually be used as-is in simple cases but should generally be adapted as required to match the requirements of a specific target/application.
- In some cases, different configurations/build targets will use different linker scripts.

**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 following option:

Cortex-M project:
*Tool Settings > Cross Arm GNU C/C++ Linker > Miscellaneous > Use newlib-nano (--specs=nano.specs)*

RISC-V project:
*Tool Settings > GNU RISC-V Cross C/C++ Linker > Miscellaneous > Use newlib-nano (--specs=nano.specs)*

**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:

Cortex-M project:
*Tool Settings > Cross Arm GNU C/C++ Compiler > Preprocessor > Defined symbols (-D)*
*Tool Settings > Cross Arm GNU C/C++ Compiler > Include paths (-I)* or *Include files (-include)*

RISC-V project:
*Tool Settings > GNU RISC-V Cross C/C++ Compiler > Preprocessor > Defined symbols (-D)*
*Tool Settings > GNU RISC-V Cross C/C++ Compiler > Include paths (-I)* or *Include files (-include)*

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

Cortex-M project:
*Tool Settings > Cross Arm GNU C/C++ Compiler > Optimization*

RISC-V project:
*Tool Settings > GNU RISC-V Cross C/C++ Compiler > Optimization*

“Fine grained” linking using `-fdata-sections -ffunction-sections` and `-gc-sections` is enabled by default here and under

Cortex-M project:
*Tool Settings > Cross Arm GNU C/C++ Linker > General > Remove unused sections (-Xlinker --gc-sections)*.

RISC-V project:
*Tool Settings > GNU RISC-V Cross C/C++ 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.

**Print Size**

By default, the *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

Cortex-M project:
*Properties* > *C/C++ Build* > *Settings* > *Tool Settings* > *Cross Arm GNU Print Size* > *General*

RISC-V project:
*Properties* > *C/C++ Build* > *Settings* > *Tool Settings* > *GNU RISC-V Cross 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 configurations/build targets (e.g. *Debug* and *Release*). To do this select *Configuration = [ All Configurations]* before specifying the relevant options and applying/saving them.

## RISC-V targets

**Do not use standard start files (-nostartfiles)**

For RISC-V targets this option must be checked when using the Microsemi Mi-V RISC-V HAL (Hardware Abstraction Layer) to avoid link errors:

*Project* > *Properties* > *C/C++ Build* > *Settings* > *Tool Settings* > *GNU RISC-V Cross C/C++ Linker* > *Do not use standard start files (-nostartfiles)*

**Use strict alignment**

Generally, and particularly when using the Microsemi RISC-V Hardware Abstraction Layer (HAL) strict alignment should be used to avoid unaligned memory access exceptions when the program is optimized for size using -Os.

*Project* > *Properties* > *C/C++ Build* > *Settings* > *Tool Settings* > *Align = Strict (-mstrict-align)*

## Cortex-M targets

**CMSIS**

Cortex-M projects require an additional setting 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 GNU C/C++ Compiler* > *Miscellaneous* set *Other compiler flags* = `--specs=cmsis.specs`.

From SoftConsole v5.3 onwards the `softconsole.cmd` (Windows) or `softconsole.sh` (Linux) script used to run SoftConsole configures the `SC_INSTALL_DIR` environment variable which is then referenced in the `<SoftConsole-install-dir>/arm-none-eabi-gcc/arm-none-eabi/lib/cmsis.specs` file. If SoftConsole is not run from the script (usually via a menu/desktop shortcut) or the Arm GCC tools are run from the command line or a Makefile then the `SC_INSTALL_DIR` environment variable will need to be configured appropriately otherwise the compiler will not find the toolchain CMSIS header files.

### SmartFusion2 Cortex-M3 targets

**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.

## 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 or SoftConsole v5.1 RISC-V projects generated by Libero SoC or the Firmware Catalog cannot be used with SoftConsole v5.2 or later.

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`, `pse_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

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. Use the example projects and debug launch configurations as a guide to creating new debug launch configurations.

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. For PolarFire SoC Renode emulation please refer to the README provided with the PolarFire SoC/PSE example project(s) in the example workspace.

4. 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:

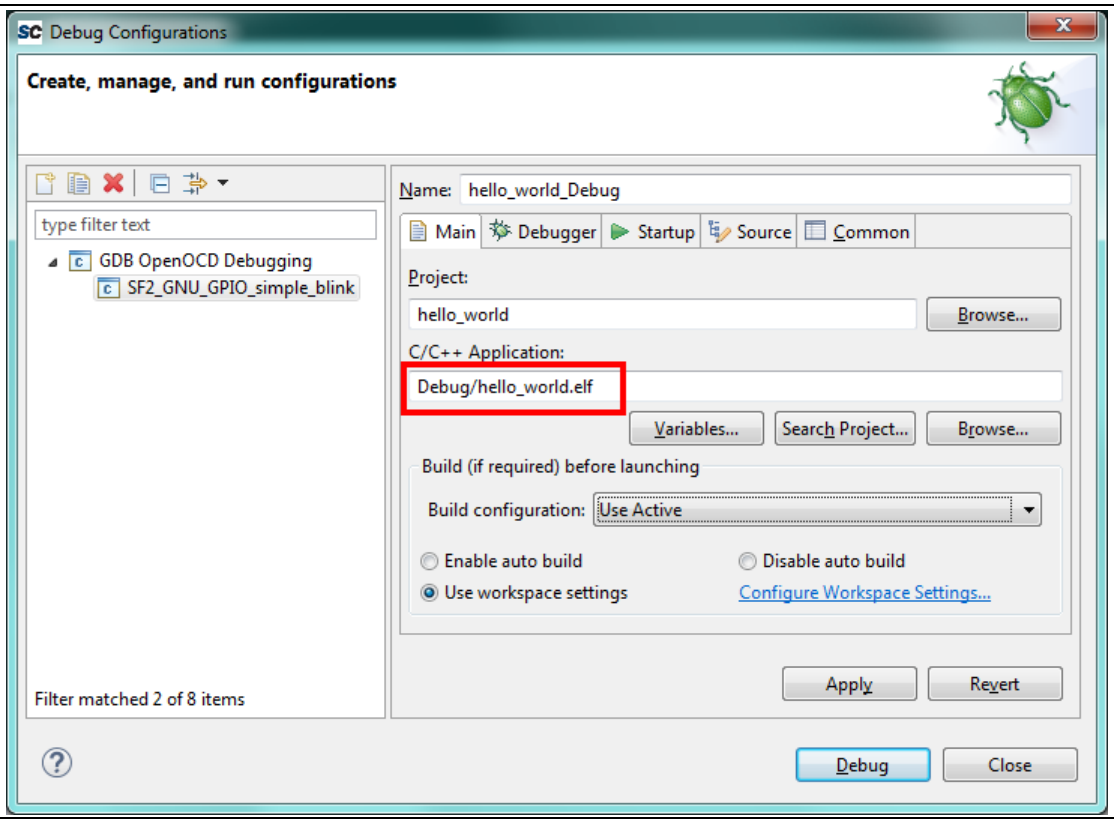



**Figure 2. Debug launch configuration Main tab**

5. 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 – if 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.

   `--command "set DEVICE ..."` is mandatory for SmartFusion and SmartFusion2 Cortex-M3 targets but is optional for Cortex-M1 and Mi-V RISC-V targets.

   For a Cortex-M1 target the *Config options* should be:

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

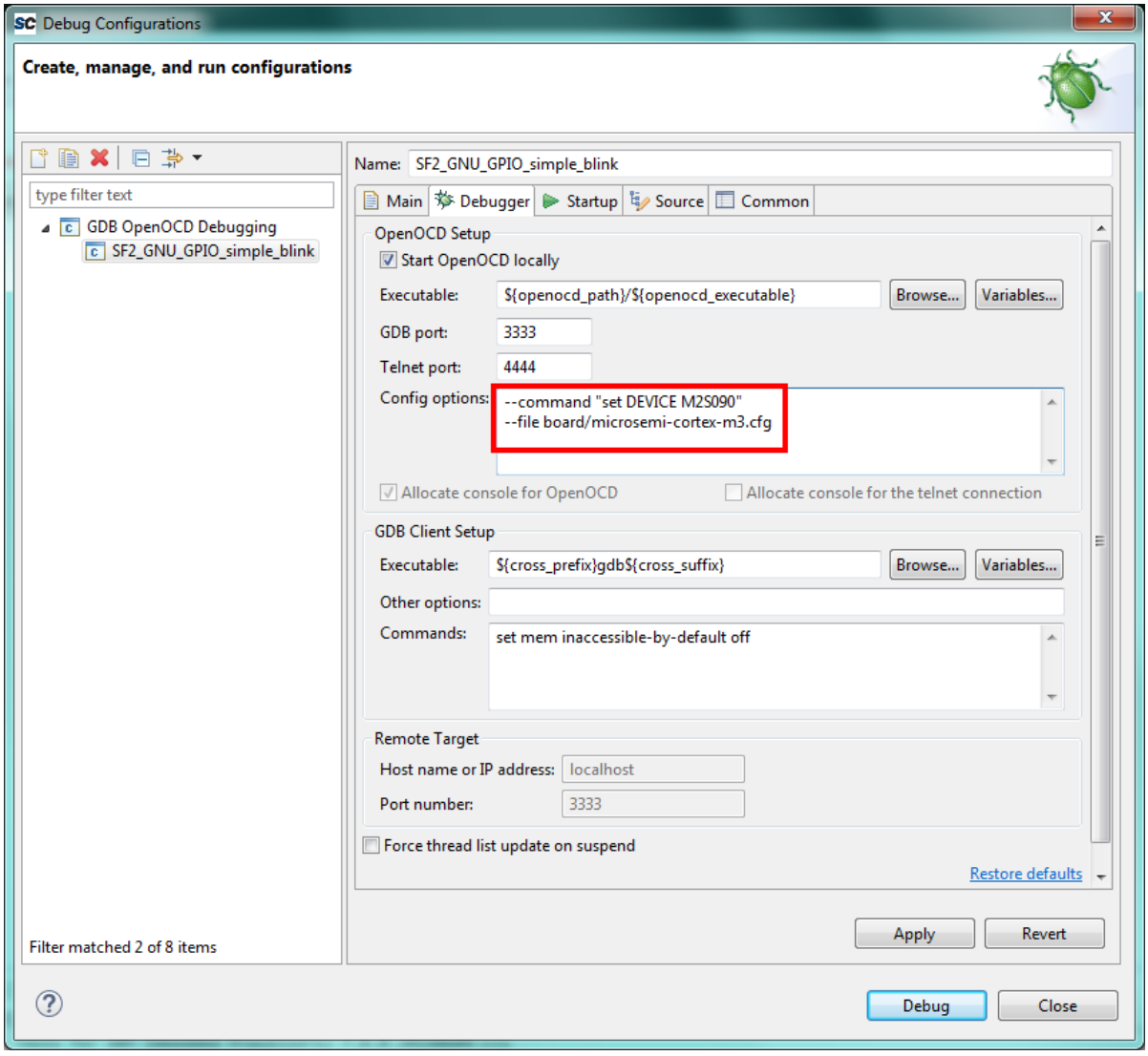



**Figure 3. Debug Configuration Debugger tab for Cortex-M3**

6. For a RISC-V target the Debugger tab settings must be configured as follows:

*OpenOCD Setup > Config options*:

```
--file board/microsemi-riscv.cfg
```

or when targeting the HiFive Unleashed Platform using the integrated FTDI JTAG debug probe (rather than FlashPro):

```
--file board/microsemi-sifive-hifive-unleashed.cfg
```

*GDB Client Setup > Commands*:

```
set mem inaccessible-by-default off
set $target_riscv = 1
```

When debugging a program on the Renode RISC-V 32-bit emulation model the bit-size of the target must be specified as follows:

```
set arch riscv:rv32
```

This is not needed where the target is actual hardware or when debugging a program on the Renode RISC-V 64-bit PolarFire SoC emulation model.

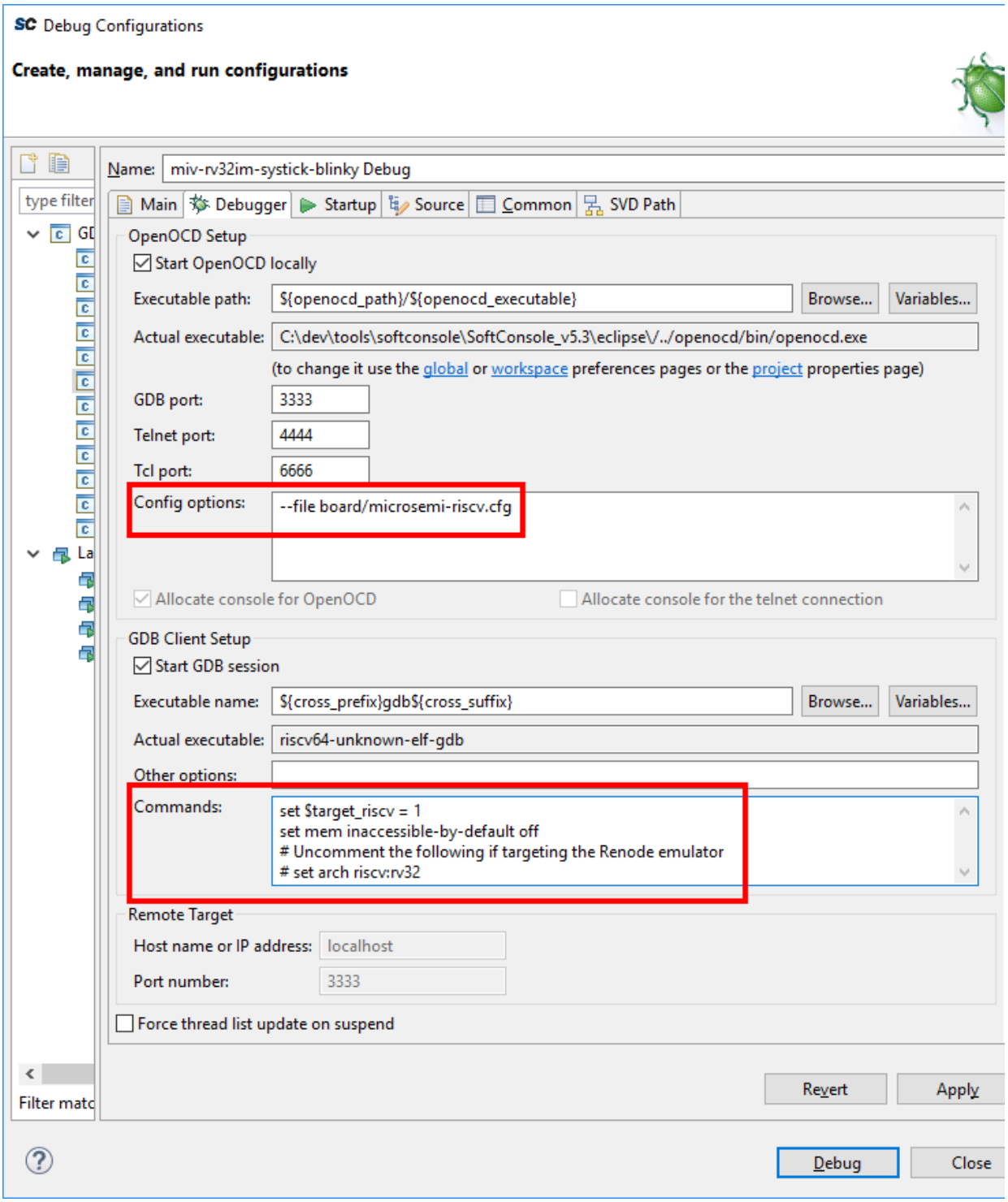



**Figure 4. Debug Configuration Debugger tab for RISC-V**

7. 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 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 semi-hosting will be used or not.

*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.

For PolarFire SoC Renode emulation please refer to the README provided with the PolarFire SoC/PSE example project(s) in the example workspace.

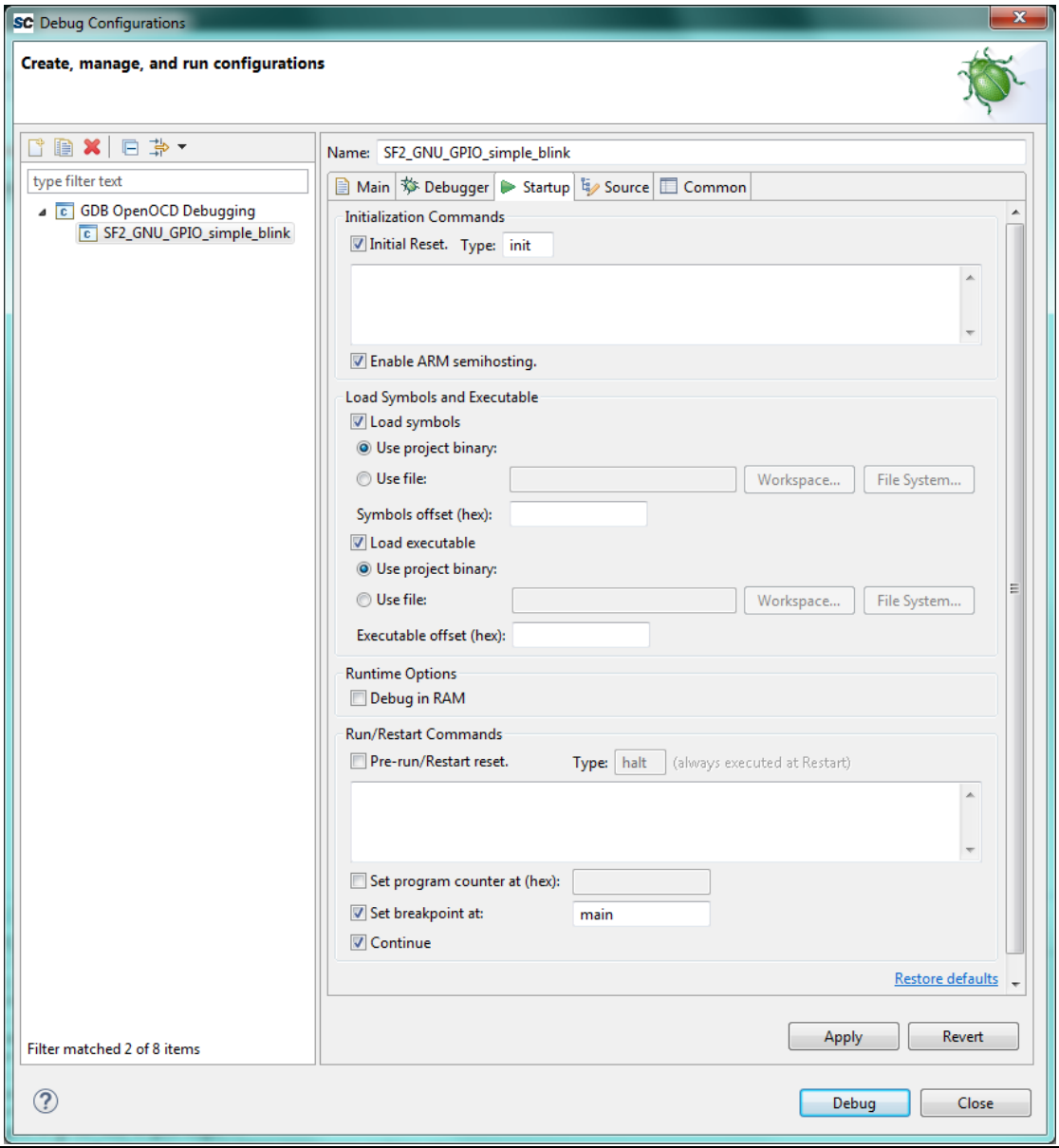



**Figure 5. Debug launch configuration Startup tab**

8. 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).

`

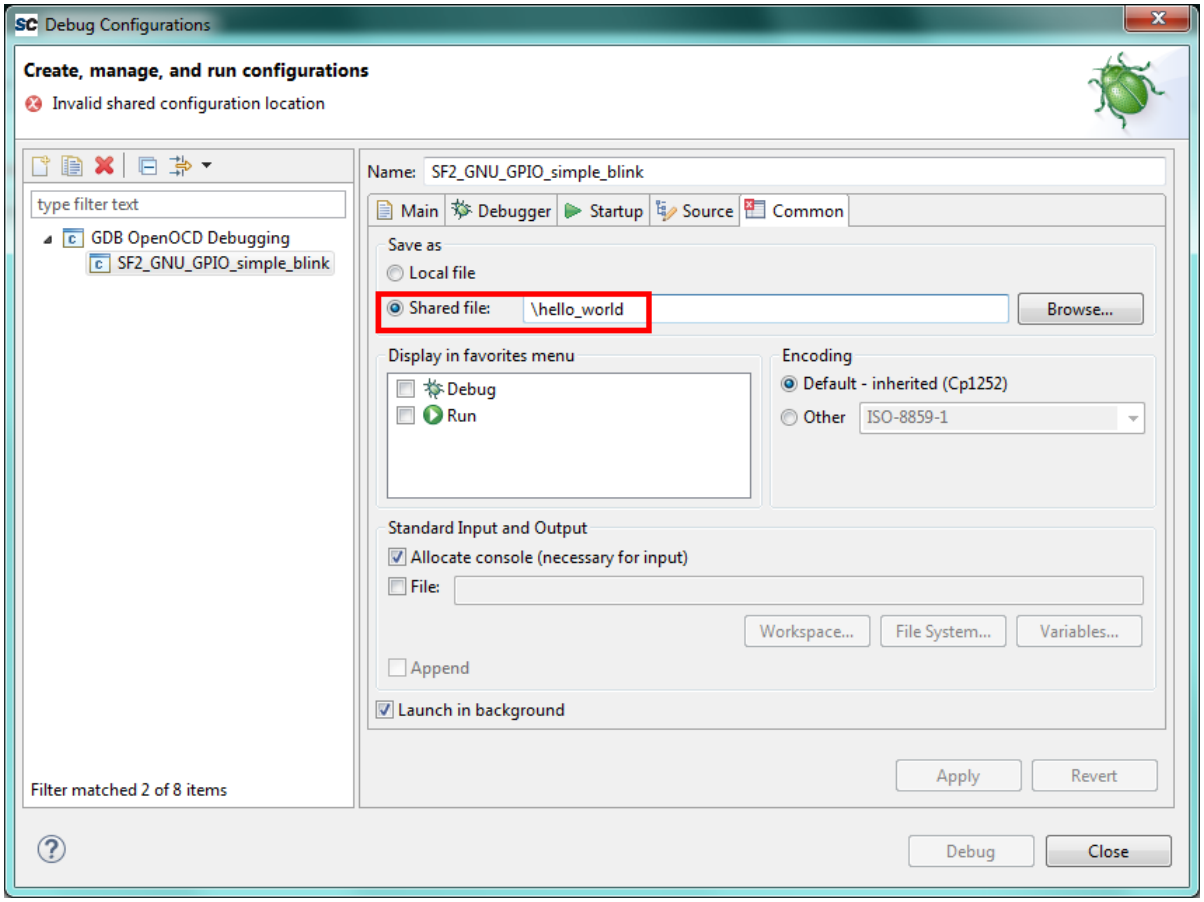



**Figure 6. Debug launch configuration Common tab**

## 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.
  - `board/microsemi-sifive-hifive-unleashed.cfg`: for targeting the HiFive Unleashed Platform

Note: For more information about SmartFusion2 MDDR external RAM support see elsewhere in this document and 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:

| Mi-V RISC-V HAL and PolarFire SoC PSE_HAL | |
|---|---|
| **Linker script** | **OpenOCD board script** |
| Refer to the Mi-V RISC-V HAL, the PolarFire SoC PSE_HAL and the various RISC-V example projects provided for details of the example linker scripts provided. | board/microsemi-riscv.cfg<br>board/Microsemi-sifive-hifive-unleashed.cfg |
| **SmartFusion2 CMSIS Hardware Abstraction Layer** | |
| **Linker script** | **OpenOCD board script** |
| debug-in-microsemi-smartfusion2-esram.ld<br>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<br>board/microsemi-smartfusion2-dev-kit-ddr.cfg<br>board/microsemi-smartfusion2-dev-kit-ddr-ecc.cfg |
| production-smartfusion2-execute-in-place.ld<br>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<br>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<br>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<br>boot-from-nvm.ld | Not applicable – production flow, not for interactive debugging |
| run-from-nvm.ld<br>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 Mi-V 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 required.

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) or simply `-d` option 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

Several 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`
- `<SoftConsole-install-dir>/openocd/share/openocd/scripts/target/microsemi-sifive-hifive-unleashed.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 several different consoles in the *Console* view. By default, the OpenOCD console is displayed showing OpenOCD output:

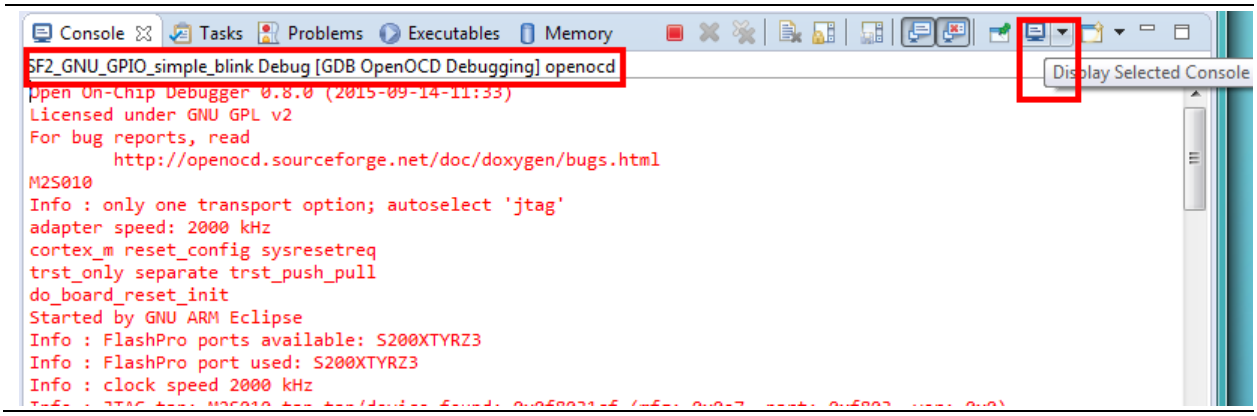



**Figure 7. Debug session – OpenOCD console view**

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

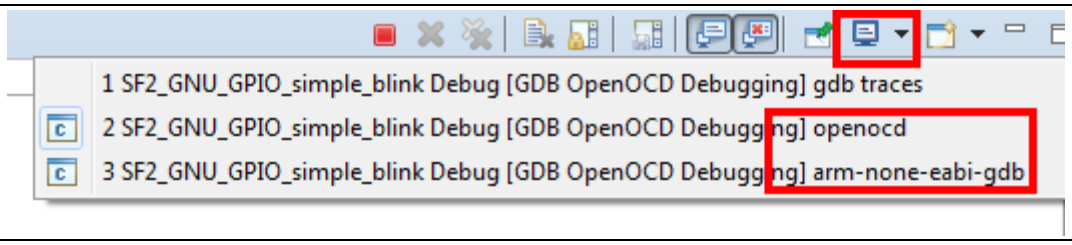



**Figure 8. Debug session – selecting a specific console view**

The *openocd* and *GDB* consoles are usually the ones of most interest. If semihosting is used the I/O is done via the GDB console. The *GDB* console must be the active console 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):
https://mcuoneclipse.com/2017/10/07/using-serial-terminal-and-com-support-in-eclipse-oxygen-and-neon/

In order for the serial terminal to list the relevant serial/COM ports, especially for USB serial ports, the relevant OS drivers may need to be installed. Refer to the relevant hardware/board documentation for more details.

## 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 and not used by any other application then SoftConsole will automatically use that. 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
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"
```

A partial port name can be specified and the first FlashPro port matched that starts with the specified string will be used. (The string comparison is case insensitive). This is useful, for example, where there are two FlashPro5 programmers attached – one standalone (e.g. SXXXXX) and one embedded e.g. EXXXXX). In this case the embedded one can be selected by simply specifying:

```
...
--command "microsemi_flashpro port e"
```

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 JTAG probe that OpenOCD supports. 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. 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 and activated in order 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 several 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` or `fpserver.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.

# Renode emulation platform

Renode™ is an open source software development framework with commercial support from Antmicro that lets you develop, debug and test multi-node device systems reliably, scalably and effectively. If all installation dependencies are met (see installation section), then the RISC-V examples can be run in the emulator.

To debug an example in the emulator Renode must be run with Mi-V model (external tool launcher called *Mi-V-Renode-emulation-platform*) or PolarFire SoC PSE model and a debug launcher attaches to the running Renode (for example *miv-rv32im-systick-blinky Attach-to-Renode*). The debug launchers for Renode are almost identical to the launchers for real 32-bit RISC-V targets except:

1. *Debugger* > *OpenOCD Setup* > *Start OpenOCD locally* has to be disabled/unchecked as Renode debugging does not use OpenOCD/JTAG but connects GDB directly to the Renode GDB Remote Serial Protocol interface.
2. RISC-V 32-bit targets need the following: *Debugger* > *GDB Client Setup* > *Commands* must specify the command `set arch riscv:rv32`

For convenience SoftConsole examples include Launch Groups that start the Renode emulation and then attach the debugger to it:

| **Launch Group name** |
|---|
| miv-rv32im-interrupt-blinky Start-Renode-emulator-and-attach |
| miv-rv32im-systick-blinky Start-Renode-emulator-and-attach |
| miv-rv32imaf-mandelbrot-uart Start-Renode-emulator-and-attach |
| miv-rv32imaf-raytracer-uart-cpp Start-Renode-emulator-and-attach |
| pse-blinky Start-Renode-emulator-and-attach |

With current launchers and models only one instance of Renode is supported, which means that the previous session needs to be completely closed before running a new session. This can be achieved either by selecting the group launcher and in the *Debug* windows and terminating it with red stop icon (Ctrl+F2) in the main toolbar on the top to stop all nested launchers. Or by closing all relevant windows from the taskbar. Failing to close all previous sessions will result failing to communicate on the port 3333 (or 3333-3337 as *pse-blinky* uses five GDB ports, one for each hart). *Remove All Terminated launchers* from the *Debug* window with the gray X icon can be used to make sure all previous sessions are terminated. Group launchers will work successfully only when the *Debug* window is showing no running launcher.

Because Renode is work-in-progress there still might be some warnings about unimplemented commands. If required the verbosity of Renode can be turned down (see the Renode documentation), but this is not recommended as it is good to see if something important is not implemented. For example, `qL12000000` warnings are only saying that the SoftConsole tries to pool thread information about RTOS tasks (even on bare-metal projects which do not use RTOS) and Renode at the moment cannot respond to these commands.

If the UART examples do not display correctly, then resize/maximizing the terminal window could resolve the issue, as the *miv-rv32iamf* examples require bigger terminal than the default. Some examples might finish very quickly, therefore for the best experience resizing should be done before continuing in the debug session.

For the PolarFire SoC example (*pse-blinky*) read the `pse-blinky/README.md` as it requires more careful considerations when creating and using launchers.

For more information see the Renode documentation located at:

- <SoftConsole-install-dir>/documentation/renode.pdf
- https://media.readthedocs.org/pdf/renode/latest/renode.pdf

- https://renode.readthedocs.io/en/latest/

# Other Features

## Cortex-M semi-hosting

Semi-hosting 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. To use semi-hosting a number of steps must be taken:

- Under *Project > Properties > C/C++ Build > Settings > Tool Settings > Cross Arm GNU C/C++ Linker > Miscellaneous > Other linker flags* add `--specs=rdimon.specs` in order to link the libraries required for semi-hosting.
- The file `CMSIS/startup_gcc/newlib_stubs.c` clashes with the semi-hosting 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 semi-hosting 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, semi-hosting 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 semi-hosting 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 GNU 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://mcuoneclipse.com/2015/08/21/gnu-static-stack-usage-analysis/

## Cppcheck

Cppcheck is a C/C++ source code static analysis tool and cppcheclipse is an Eclipse plugin that integrates cppcheck into the Eclipse/CDT SoftConsole IDE. Using cppcheck/cppcheclipse projects can be scanned for common source code issues and bugs each time the project is built or on demand. The configuration options for cppcheck/cppcheclipse are accessible via:

*Project Properties > cppchecklipse*

For more information on using cppcheck/cppcheclipse source code static analysis capabilities refer to these links:

- https://mcuoneclipse.com/2015/07/02/open-source-static-code-analysis-cppcheck-with-eclipse/
- https://github.com/kwin/cppcheclipse/wiki/WorkspacePreferences
- http://cppcheck.sourceforge.net/
- http://cppcheck.sourceforge.net/manual.pdf
- https://sourceforge.net/projects/cppcheck/files/Articles/

cppcheck can also be used from command line or as part of a Makefile based project. The required cppcheck binaries are located in `<SoftConsole-install-dir>/extras/cppcheck`.

# Known Issues and useful tips

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.

## Reset/power cycle the target hardware before each RISC-V debug session

At the moment, the debugger cannot effect a suitable RISC-V CPU/SoC reset at the start of each debug session so one debug session may be impacted by what went before – e.g. a previous debug session leaves the CPU in an ISR and a subsequent debug session does not behave as expected because of this. To mitigate this problem, it is recommended that the target hardware/board is power cycled or otherwise reset before each new debug session.

## 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.

## RISC-V memory view problems

When using the Memory Monitor or Memory Browser views to view memory in a Mi-V RISC-V system warnings such as the following may appear in the debug/OpenOCD log view – these can be ignored for the moment.

```
Warn : negative acknowledgment, but no packet pending
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent!
(1001). Workaround: increase "set remotetimeout" in GDB

or

Info : dtmcontrol_idle=5, dmi_busy_delay=8278771, ac_busy_delay=0
```

In some cases, the memory view may not display the memory contents correctly displaying, instead, question marks. This will be fixed in a future release.

## 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.

## 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

RISC-V C++ projects have not been extensively tested within the SoftConsole Eclipse/CDT environment. The underlying RISC-V GNU toolchain does support C++ but the SoftConsole IDE may not yet properly support C++ projects.

## FlashPro programmers cannot be shared by applications

Due to limitations of the FlashPro driver/library software support used by FlashPro client applications (e.g. SoftConsole OpenOCD, SmartDebug, Identify etc.) FlashPro programmers cannot be shared and used at the same time by multiple applications and only one application can use a specific FlashPro programmer at any one time.

## 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 while the projects in the example workspace are indexed. 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

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, `fpServer.exe` and/or Renode. Allow the firewall to unblock these and save this as the default setting if necessary.

## Multiple debug sessions

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

## 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.

## “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), 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 may display a message of the form:

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

Such error messages can be safely ignored.

## OpenOCD error/info messages when debugging RISC-V

When debugging a RISC-V target the following error/info messages may appear but the debug session proceeds without problems. These messages can be safely ignored for now.

```
Info : RISC-V IDCODE = 0x10e31913
Info : dtmcontrol_idle=5, dmi_busy_delay=1, ac_busy_delay=0
Info : dtmcontrol_idle=5, dmi_busy_delay=2, ac_busy_delay=0
Info : dtmcontrol_idle=5, dmi_busy_delay=3, ac_busy_delay=0
Info : dtmcontrol_idle=5, dmi_busy_delay=4, ac_busy_delay=0
Error: Unable to execute program 0123ed74
Info : Disabling abstract command reads from CSRs.

...

Info : accepting 'gdb' connection on tcp/3333
Error: Unable to execute program 0123f554
Error: failed to execute program, abstractcs=0x0e000001
Error:   exiting with ERROR_FAIL

...
```

## Debugging and multiple device JTAG chains

Debugging a particular SmartFusion or SmartFusion2 Cortex-M3 in a multiple device JTAG 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-cortex-m3.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 {} {
}
```

Debugging a particular UJTAG/CoreJTAGDebug connected Cortex-M1 or RISC-V in a multiple device JTAG chain is not yet possible.

Where there are multiple UJTAG/CoreJTAGDebug connected Cortex-M1 and/or Mi-V RISC-V CPUs in a single device it is possible to debug any one of these at a time by specifying the appropriate IRCODE configured in CoreJTAGDebug for the relevant CPU. E.g.:

```
--command "set UJ_JTAG_IRCODE 0x34" --file board/microsemmi-cortex-m1.cfg
```

or

```
--command "set UJ_JTAG_IRCODE 0x56" --file board/microsemmi-riscv.cfg
```

The default UJ_JTAG_IRCODE used by `target/microsemi-cortex-m1.cfg` is 0x33 and by `target/microsemi-riscv.cfg` is 0x55.

## RISC-V target support

SoftConsole supports software development and debug for Microsemi Mi-V 32-bit soft cores and PolarFire SoC 64-bit multiprocessor targets and comes bundled with the set of multilibs (for specific architecture/abi configurations) detailed earlier in the document. However it should be possible to develop and debug with any other RISC-V implementation that is covered by the bundled multilibs and which adheres to the RISC-V User-Level ISA (Instruction Set Architecture) v2.2 (https://riscv.org/specifications/) and the RISC-V Draft Privileged ISA Specification v1.10 (https://riscv.org/specifications/privileged-isa/).

SoftConsole and the underlying RISC-V GCC development/debug tools are configured to use RISC-V Draft Privileged ISA Specification v1.10 names and locations for CSRs (Control and Status Registers) so may not work correctly for RISC-V implementations that adhere to any earlier draft version of that specification.

## 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 or later. 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 or later. Note that SoftConsole v5.1 and later 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.

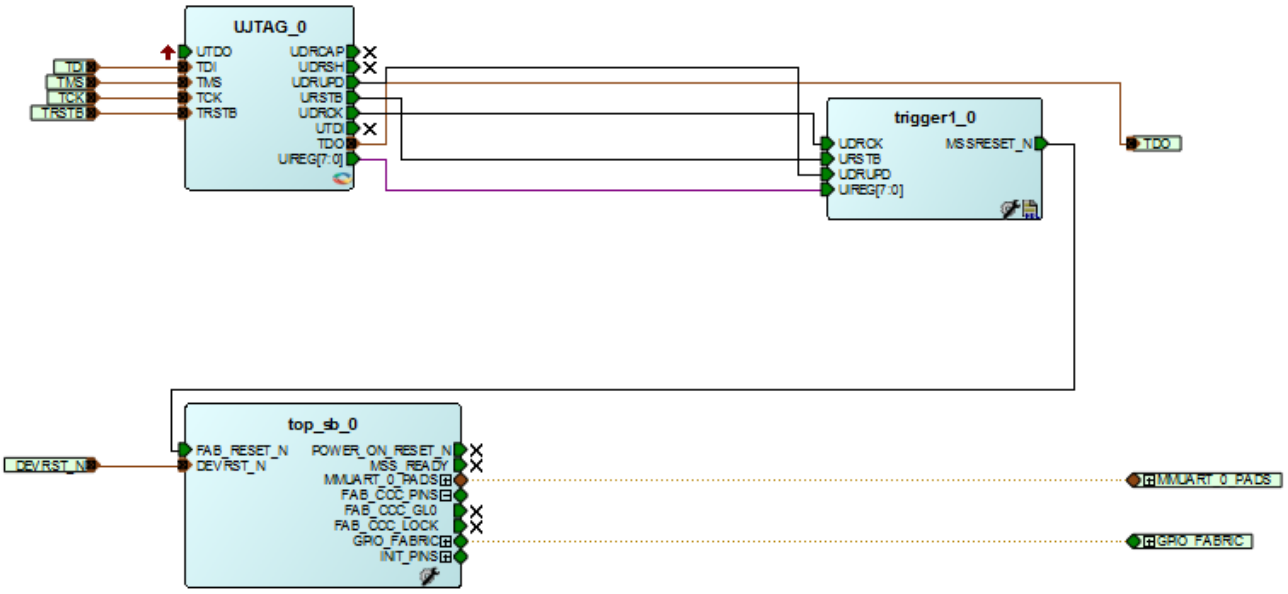
## SmartFusion2 DPK unlocking

SmartFusion2 provides an option to lock down Cortex-M3 debug access using a DPK (Debug PassKey). If this is enabled in the Libero design, then SoftConsole/OpenOCD needs to specify the DPK for debugging to work. To do this the 256-bit DPK must be passed to OpenOCD as a 64-hex digit string as follows:

```
--command "set DPK 0x0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF"
--command "set DEVICE M2S090"
```

```
--file board/microsemi-cortex-m3.cfg
```

There is a known issue with some or all SmartFusion2 devices whereby the DPK unlock only works if the MSS is reset after the DPK unlock operation has been executed and this does not happen automatically. If debug access does not work even though the correct DPK was specified as above, then it may be necessary to add some logic to the FPGA design to reset the MSS on detection of the DPK unlock operation at the UJTAG level. This logic and Verilog implementation is outlined below.




```
module trigger1( UDRCK, URSTB, UDRUPD, MSSRESET_N, UIREG );
input UDRCK, URSTB, UDRUPD;
output MSSRESET_N;
input [7:0] UIREG;

reg MSSRESET_N;
reg [1:0] state;

parameter  CHECK = 2'b01, HIGH = 2'b10;

always @ (posedge UDRCK or negedge URSTB)
    begin
        if (!URSTB)
        begin
            MSSRESET_N <= 1;
            state <= CHECK;
        end
        else
        begin
            case(state)
                CHECK:
                begin
```

```
                    if ((UIREG[7:0] == 8'h0A) && (UDRUPD == 1))
                    begin
                        MSSRESET_N <= 0;
                        state <= HIGH;
                    end
                    else
                    begin
                        MSSRESET_N <= 1;
                        state <= CHECK;
                    end
                end

                HIGH:
                begin
                    MSSRESET_N <= 1;
                    state <= CHECK;
                end

                default:
                begin
                    MSSRESET_N <= 1;
                    state <= CHECK;
                end
            endcase
        end
    end
endmodule
```

## CMSIS needs environment variable

If an ARM project is built with a Makefile or any other unmanaged process then the SoftConsole install location needs to be exported for the CMSIS to work:

```
export SC_INSTALL_DIR=<INSTALL_PATH>
```

When SoftConsole is run from the provided launcher script (softconsole.cmd on Windows and softconsole.sh on Linux) then the script configures this environment variable so that the Arm GCC toolchain can find the CMSIS toolchain header files. If, when compiling a Cortex-M project, the CMSIS toolchain header files are not found then make sure that SoftConsole is launched via the script or that the `SC_INSTALL_DIR` environment variable is configured correctly on the system. In the past SoftConsole could be run by simply running `<SoftConsole-install-dir>/eclipse/eclipse.exe` but if this is done in SoftConsole v6.0 or later then the `SC_INSTALL_DIR` environment variable will not be set by default.

## FTDI detected as ttyUSB on Linux

In some instances, all the FTDI channels might get taken by the serial driver and turned into the ttyUSB devices. If this happens then OpenOCD may list one or more FlashPro programmers but fail to connect to one:

```
Info : FlashPro ports available: S201Z7LB20
Info : FlashPro port used: S201Z7LB20
Error: InitializeProgrammer(S201Z7LB20) failed : Can not connect to the programmer
```

Running `dmesg` can show that the `ftdi_sio` driver is registered to the device.

```
[ 1400.220327] usb 2-1.4: Product: FlashPro5
[ 1400.220329] usb 2-1.4: Manufacturer: Microsemi
[ 1400.220331] usb 2-1.4: SerialNumber: 01Z7LB20
[ 1400.280943] usbcore: registered new interface driver ftdi_sio
[ 1400.280972] usbserial: USB Serial support registered for FTDI USB Serial Device
[ 1400.281176] ftdi_sio 2-1.4:1.2: FTDI USB Serial Device converter detected
[ 1400.281264] usb 2-1.4: Detected FT4232H
[ 1400.281726] usb 2-1.4: FTDI USB Serial Device converter now attached to ttyUSB0
```

A script can be used as a temporary workaround which will then disable all serial drivers with the FTDI. This script will unload the `ftdi_sio` module:

```
#!/bin/sh
# $DEVNAME enviroment variable is most likely set to /dev/ttyUSB0
# get the ttyUSB0 without the path
NAME=$(basename $DEVNAME)

# Find FTDI child USB device with then $DEVNAME parent device
for DEVICE in /sys/bus/usb/drivers/ftdi_sio/*/
do
    if [ -e $DEVICE/$NAME ]
    then
        # Match found, unbind it from the FTDI driver
        echo -n "$(basename $DEVICE)" > /sys/bus/usb/drivers/ftdi_sio/unbind
        break
    fi
done
```

Save this script as `/usr/local/bin/unbind_ftdi.sh` and give it correct privileges:

```
sudo chmod a+x /usr/local/bin/unbind_ftdi.sh
sudo chown root:staff /usr/local/bin/unbind_ftdi.sh
```

Now find where the OpenOCD udev script was installed (most likely `/etc/udev/rules.d/60-openocd.rules`) and modify the existing *"Microsemi (Actel) FlashPro5"* entry as follows:

```
# Microsemi (Actel) FlashPro5
ATTRS{idVendor}=="1514", ATTRS{idProduct}=="2008", MODE="666",
RUN+="/usr/local/bin/unbind_ftdi.sh"
```

Depending on the distribution I might be necessary to reload the udev rules or reboot for the changes to take effect:

```
udevadm trigger
```

Now reconnecting the FlashPro device and watching `dmesg` messages should state that the `ftdi_sio` module was unloaded and OpenOCD should be able to connect to and use the connected FlashPro programmer(s).

```
[10626.270300] usb 2-1.4: new high-speed USB device number 54 using ehci-pci
[10626.382547] usb 2-1.4: New USB device found, idVendor=1514, idProduct=2008
[10626.382554] usb 2-1.4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[10626.382556] usb 2-1.4: Product: FlashPro5
[10626.382560] usb 2-1.4: Manufacturer: Microsemi
[10626.382562] usb 2-1.4: SerialNumber: 01Z7LB20
[10626.387020] ftdi_sio 2-1.4:1.2: FTDI USB Serial Device converter detected
[10626.387085] usb 2-1.4: Detected FT4232H
[10626.387615] usb 2-1.4: FTDI USB Serial Device converter now attached to ttyUSB0
[10626.432272] ftdi_sio ttyUSB0: FTDI USB Serial Device converter now disconnected from
ttyUSB0
[10626.432296] ftdi_sio 2-1.4:1.2: device disconnected
```

## Program file does not exist

Possible cause: If a build finishes successfully and the ELF file exists but the debug launch configuration cannot see it, then a likely cause is the use of the wrong slash in the program path name. It is advisable to use the forward slash (/) in such paths for Windows and Linux cross OS compatibility. If the backslash (\) is used then it will only work on Windows but the forward slash works on both. Unfortunately the debug launch configuration editor on Windows defaults to using the backslash.

Workaround: Change the slash in launcher configuration C/C++ Application setting:

e.g. from "Debug\blinky.elf" to "Debug/blinky.elf"

## Target emulation `elf64-littleriscv' does not match `elf32-littleriscv'

Possible cause: This can be caused when selected RISC-V architecture/abi does not have a matching multilib and the default RV64GC is used instead.

Workaround: Use one of the arch/abi combinations mentioned earlier as being supported with a corresponding multilib.

## sprint()/scanf() print empty characters instead of floating point number digits

Possible cause: Support in the libraries is not enabled by default when newlib-nano is used by having the *Project Settings > C/C++ Build > Settings > Tool Settings > GNU RISC-V Cross C/C++ Linker > Miscellaneous > Use newlib-nano (--specs=nano.specs)* checked.

Workaround: Check the *Project Settings > C/C++ Build > Settings > Tool Settings > GNU RISC-V Cross C/C++ Linker > Miscellaneous > -u_printf_float* and/or *-u_scanf_float* options. If newlib-nano is NOT in use but this problem occurs then it may be caused by the GNU or a third party standard library used instead of newlib.

## Error message: can't link hard-float modules with soft-float modules

Possible cause: Project settings affecting float usage changed without doing a clean rebuild.

Workaround: Delete the *Debug/Release* folders and check that the RVF floating point configuration options and selected arch/abi match. Then rebuild the program from scratch.

## Multiple definition of `_start'/`_init'/`_fini' etc.

Errors of the following form appear:

```
multiple definition of `_start'
multiple definition of `_init'
multiple definition of `_fini'
```

Possible cause: Trying to the RISC-V or PolarFire SoC startup code while the compiler is trying to use its own startup code.

Workaround: Do not use the compiler startup code. Check the *Project Settings > C/C++ Build > Settings > Tool Settings > GNU RISC-V Cross C/C++ linker > General > -nostartfiles* option

## Undefined reference to `printf'/`puts'/`write'/`strlen' etc.

Errors of the following form appear:

```
undefined reference to `printf'
undefined reference to `puts'
undefined reference to `write'
undefined reference to `strlen'
```

Possible cause: No libraries are present to implement these calls.

Workaround: Uncheck the *Project Settings > C/C++ Build > Settings > Tool Settings > GNU RISC-V Cross C/C++ Linker > General > Do not use default libraries (-nodefaultlibs)* and *No startup or default libs (-nostdlib)* options.

## Route `printf()` output to UART

To route printf() output to UART:

1. Set up CoreUART in such way that `UART_polled_tx_string()` output works
2. Add the following include file:

   ```
   #include <stdio.h>
   ```

3. Add the define symbol "**MSCC_STDIO_THRU_CORE_UART_APB**" to In *Project Settings > C/C++ Build > Settings > Tool Settings > GNU RISC-V Cross C/C++ Compiler > Preprocessor > Defined Symbols* add the symbol `MSCC_STDIO_THRU_CORE_UART_APB`.

## Program image is too large

If the program image size is too large, then check the list and map files for details of what application and library code is contributing to the total size. Note that the ELF file size on disk is not indicative of the program image size when running on the embedded hardware target. Some useful common tips for reducing the program image size:

1. Tell the compiler to optimize: *Project Settings > C/C++ Build > Settings > Tool Settings > Optimization > Optimization Level = Optimize for size -Os* (this will negatively affect debugging capabilities)

2. Use newlib-nano: Check the option *Project Settings > C/C++ Build > Settings > Tool Settings > GNU Arm/RISC-V Cross C/C++ Linker > Miscellaneous > Use newlib-nano (--spec=nano.specs)* and uncheck the options *Project Settings > C/C++ Build > Settings > Tool Settings > GNU Arm/RISC-V Cross C/C++ Linker > General > Do not use default libraries (-nodefaultlibs) and No use startup or default libs (-nostdlib).*

3. If `printf()` style functions are being used and the program is still too large even when using newlib-nano then it may be possible to use a standalone small `printf()` implementation instead. Most such implementations may have limitations compared to a "full" standard library `printf()` implementation but often these limitations are acceptable in an embedded context. Examples of such "small" `printf()` implementations include:

   https://github.com/mludvig/mini-printf

   http://www.xappsoftware.com/wordpress/2011/01/17/a-tiny-printf-for-embedded-systems/

   https://github.com/mpaland/printf (minimalistic `printf()` that supports floats)

   Then disable the use of the compiler standard libraries:

   Check the option *Project Settings > C/C++ Build > Settings > Tool Settings >GNU Arm/RISC-V Cross C/C++ Linker > General > Do not use default libraries (-nodefaultlibs)*.

   Better still consider rewriting the relevant code to avoid the use of `printf()` functions which are generally not recommended on embedded platforms because they are often large and often use the heap.

4. Wrap code that is not always required (e.g. verbose logging code used for debugging/diagnostics only) in `#ifdef <symbol> ... #endif` blocks so that they can be conditionally compiled out when not needed.

5. Check if the floating point is using hardware and not software implementation. Not enabling hardware properly and using floating point will emulate every instruction in software and enlarge the codebase significantly. Do not use RVF or RVD floating point without setting the corresponding floating point ABI in *Project Settings > C/C++ Build > Settings > Target processor*. With RISC-V cores implementing the F and/or D extensions the *Floating-point divide/sqrt instructions (-mfdiv)* can also be enabled.

6. Check what function or blocks of project code should be prioritized for optimization. Create a size analysis post build step under:

   *Project Settings > C/C++ Build > Settings > Build Steps > Post-build steps*

   and put the following in the *Command* field:

   ```
   ${cross_prefix}nm${cross_suffix} --print-size --size-sort ${BuildArtifactFileName}
   ```

   Optionally a *Description* can be given to it, for example *"------- Size analysis -------"*.

   After the build all symbols with their sizes will be displayed and ordered by the size. This can point to functions which will benefit from optimizations the most. It can reveal design/code issues as when a large library could have been linked into a project by mistake or other unnecessary code compiled in.

7. Enable hardware features if the core is capable – e.g. use the RISC-V M extension fully if present. *Under Project Settings > C/C++ Build > Settings > Tool Settings > Target processor* enable/check the *Multiply extension (RVM)* and the *Integer divide instructions (-mdiv)* options if appropriate. If the target implements the *Compressed extension (RVC)* then enable/check that too:

## cc1: error: requested ABI requires -march to subsume the 'D' extension

Or the following:

```
cc1: error: ABI requires -march=rv64
```

Possible cause: The RISC-V *Target Processor* > *Toolchain* option was left configured as *Toolchain default* which will result in the compiler assuming that the target arch/abi is rv64gc/lp64.

Workaround: Ensure that the *Project Settings* > *C/C++ Build* > *Settings* > *Target processor* > *Architecture, Integer ABI* and *Floating point ABI* are configured correctly for the target hardware.

## undefined reference to `__stack_top' etc.

Or the following errors:

```
undefined reference to `__sdata_start'
undefined reference to `__data_start'
undefined reference to `__sbss_start'
undefined reference to `_heap_end'
undefined reference to `__dso_handle'
hidden symbol `__dso_handle' isn't defined
```

Possible cause: Using old/wrong/no linker script.

Workaround: Ensure that the appropriate linker script is configured under *Project Settings* > *C/C++ Build* > *Settings* > *Tool Settings* > *GNU Arm/RISC-V Cross C/C++ Linker* > *Add linker script*.

## After duplication of the project (copy/paste) debugging no longer works

Possible cause: The copy/paste renames everything except launch configurations.

Workaround: Update the path to ELF file under:

*Run* > *Debug configurations…* > *<PROJECT_DEBUG_LAUNCHER>* >*C/C++ Application*

## Progam has exited with code:0x00000003

Or any other exit codes.

Possible cause: The code got into unhandled exception/trap state

Workaround: Troubleshoot as a trap, the RISC-V exit code meanings are:

```
0x00000001=Instruction address misaligned
0x00000002=Instruction access fault
0x00000003=Illegal instruction
0x00000004=Breakpoint
0x00000005=Load address misaligned
0x00000006=Load access fault
0x00000007=Store/AMO address misaligned
0x00000008=Store/AMO access fault
0x00000009=Environment call from U-mode
0x0000000A=Environment call from S-mode
0x0000000C=Environment call from M-mode
0x0000000D=Instruction page fault
0x0000000E=Load page fault
```

```
0x00000010=Store/AMO page fault
```

The exit code is the RISC-V Privilege Specification `mcause` register value plus 1. For full information refer to the draft RISC-V Privileged ISA Specification:

https://riscv.org/specifications/privileged-isa/

Get value of mepc CSR register, it points to memory where the trap happened. Create breakpoint on this location. Single step if needed and the moment right after the trap happened, get values from the following CSRs: mcause, mip, mie, mtval/mbadaddr, mtvec, mepc, mscratch and mstatus. When asking for assistance, all these CSRs can give vital information to troubleshoot the problem.

**mcause** Should be the same as exi code -1, if it's not then different trap happened in this debug session.

**mip** Displays pending interrupts.

**mie** Displays what interrupts are enabled.

**mtval/mbadaddr** Depending on the assembler the name of the CSR might be either mtval or mbadaddr. If the exit code is 1,2, 5, 6, 7, 8, E or 10 then this CSR points to the memory address which caused this fault. If the exit code is 3 then this CSR contains the opcode which caused it. Together with the exit code this can point to what happened and where it happened. Note that mcause and exit code are almost identical except the exit code is offset by 1.

**mtvec** Is trap handler vector, address where to jump when the trap happens.

**mepc** Is the value of Program Counter at the moment of the exception (address to code which caused the trap).

**mscratch** Temporary values, sometimes it can hold value of the A0 register which is used for arguments passing or return values from the functions.

**mstatus** Can contain global interrupt enable bit but many other flags:

| 31 | 30 … 23 | 22 | 21 | 20 | 19 | 18 | 17 |
|---|---|---|---|---|---|---|---|
| SD | WPRI | TSR | TW | TVM | MXR | SUM | MPRV |
| 1 | 8 | 1 | 1 | 1 | 1 | 1 | 1 |

| 16 15 | 14 13 | 12 11 | 10 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| XS[1:0] | FS[1:0] | MPP[1:0] | WPRI | SPP | MPIE | WPRI | SPIE | UPIE | MIE | WPRI | SIE | UIE |
| 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Machine-mode status register (`mstatus`) for RV32.

| XLEN-1 | XLEN-2 … 36 | 35 34 | 33 32 | 31 … 23 | 22 | 21 | 20 | 19 | 18 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|
| SD | WPRI | SXL[1:0] | UXL[1:0] | WPRI | TSR | TW | TVM | MXR | SUM | MPRV |
| 1 | XLEN-37 | 2 | 2 | 9 | 1 | 1 | 1 | 1 | 1 | 1 |

| 16 15 | 14 13 | 12 11 | 10 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| XS[1:0] | FS[1:0] | MPP[1:0] | WPRI | SPP | MPIE | WPRI | SPIE | UPIE | MIE | WPRI | SIE | UIE |
| 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Machine-mode status register (`mstatus`) for RV64 and RV128.

## Size optimizations cause trap exception: Load address misaligned

Possible cause: Using default HAL without any modifications and allowing unaligned accesses in the project settings.

*Fix: Project > Properties > C/C++ Build > Settings > Tool Settings > Align Strict (-mstrict-align)*

## <built-in>: internal compiler error: Illegal instruction

Possible cause: The gmp big number library which is bundled into the gcc might have been set to use a newer cpu than you actually have.

Fix: Contact us, so we know what CPUs target for the next release.

## (Renode:5523): GLib-CRITICAL **: Source ID 13802 was not found when attempting to remove it

Possibly caused by not meeting Renode's dependancies. Renode requires the mono to be at least v5. Double check if the newest versions of the mono-complete, gtk-sharp2 and libcanberra-gtk are installed. And confirm the mono is v5 or higher:

```
mono --version
```

## ../riscv_hal/entry.S:113: Error: Instruction csrr requires absolute expression

Possible cause: The RISC-V assembler is behind and is not recognizing the newer name mtval of the CSR.

Fix: Use older mbadaddr CSR name instead of the mtval which is the same CSR.

## Connecting to remote OpenOCD

The binding behavior of the OpenOCD changed from the SoftConsole 5.2. Now by default, it's binding itself to the 127.0.0.1 instead of the 0.0.0.0 which means that the remote OpenOCD will listen only to its loopback address instead on all interfaces and it will not be reachable remotely. This default behavior can be overridden by adding the **-c "bindto 0.0.0.0"** argument to the OpenOCD command.

```
openocd.exe -c "bindto 0.0.0.0" <rest-of-your-arguments>
```

## Glitches, unstable UI and cosmetic issues on Linux

On the Linux SoftConsole 6.0 the GTK2 is getting deprecated and it might cause incorrectly rendered UI elements in some cases and on some distributions even unstable UI. If SoftConsole 6.0 is not used for ARM projects then it's safe to comment out with a # symbol (or remove) the line below from the **softconsole.sh** launcher:

```
export SWT_GTK3=0 # force GTK 2 because: "ARM project settings -> target" was extremely slow with GTK3
```

## Debugger Console View is not working or GDB is misbehaving

SoftConsole 6.0 debugger version detection can fail and not show Debugger Console. The Debugger Console will work when macros are not used, copy content of the **Debug Configuration -> Debugger -> GDB Client Setup -> Actual name** (should be "riscv64-unknown-elf-gdb" or "arm-none-eabi-gdb") and then paste it instead of the **${cross_prefix}gdb${cross_suffix}** macro in the **Debug Configuration -> Debugger -> GDB Client Setup -> Executable**.

## Build fails when using “Print removed sections (-Xlinker --print-gc-sections)”

When the option to print removed unused sections is selected CDT may report that the build has failed even though it was completed correctly. This is because of a bug in CDT whereby the `--print-gc-sections` option prints removed sections to stderr and CDT incorrectly interprets ANY output on stderr to be a build error. This bug has been reported to the CDT project and for now it is safe to ignore such build failure false positives. Alternatively disable the `--print-gc-sections` option by unchecking the following option:

RISC-V: *Project Properties > C/C++ Build > Settings > Tool Settings > GNU RISC-V Cross C/C++ Linker > General > Print removed sections (-Xlinker --print-gc-sections)*

Arm projects: *Project Properties > C/C++ Build > Settings > Tool Settings > Cross Arm GNU C/C++ Linker > General > Print removed sections (-Xlinker --print-gc-sections)*

When the `--print-gc-sections` option is enabled then the following false positive build failure may occur:

```
riscv64-unknown-elf/bin/ld.exe: Removing unused section '.text.__disable_irq' in file
'./riscv_hal/riscv_hal.o'
riscv64-unknown-elf/bin/ld.exe: Removing unused section '.text.handle_m_ext_interrupt' in
file './riscv_hal/riscv_hal_stubs.o'
riscv64-unknown-elf/bin/ld.exe: Removing unused section '.text.SysTick_Handler' in file
'./riscv_hal/riscv_hal_stubs.o'
riscv64-unknown-elf/bin/ld.exe: Removing unused section '.sbss.__env' in file
'./riscv_hal/syscall.o'
...
riscv64-unknown-elf/bin/ld.exe: Removing unused section '.text.isatty' in file
'lib/rv32im/ilp32\libg_nano.a(lib_a-sysisatty.o)'
riscv64-unknown-elf/bin/ld.exe: Removing unused section '.debug_frame' in file
'lib/rv32im/ilp32\libg_nano.a(lib_a-sysisatty.o)'
riscv64-unknown-elf/bin/ld.exe: Removing unused section '.sdata._global_impure_ptr' in file
'lib/rv32im/ilp32\libg_nano.a(lib_a-impure.o)'
Finished building target: miv-rv32im-systick-blinky.elf

12:35:47 Build Failed. 71 errors, 0 warnings. (took 2s.428ms)
```

## Debug launcher fails with “Error: gdb sent a packet with wrong register size”

```
Error: gdb sent a packet with wrong register size
Info : dropped 'gdb' connection
```

This indicates mismatch between target architecture and what is expected. This can happen when debugging RISC-V 64-bit software on a RISC-V 32-bit target (or vice versa) and in general it is critical that the selected architecture and abi match the target platform.

close attention needs to be paid to the selected architecture and abi needs to be paid , checking project settings. If floating point extension and floating point abi is set correctly. If the debug launcher is not setting wrong arch, found in the launcher properties: **Debug Configuration -> Debugger -> GDB Client Setup -> Commands -> set arch riscv:rv32**

In an edge case this problem might be present even when all settings are correct, but using Makefile non-managed project on Linux SoftConsole. This happens when cross compiler is incorrectly detected and the native tools are used instead (which are x86 and they do not match the RISC-V registers). **Debug Configuration -> Debugger -> GDB Client Setup -> Executable name** ${cross_prefix}gdb${cross_suffix} is tool name which will be used, it uses macros and should be resolved into correct actual target name. The **Debug Configuration -> Debugger -> GDB Client Setup -> Actual name** should be **riscv64-unknown-elf-gdb**. In case this actual name got resolved into **gdb** only then the issue can be fixed by hardcoding full name **riscv64-unknown-elf-gdb** without using the ${cross_prefix} and ${cross_suffix} macros.

## Debug launcher is not connecting to ARM target

Info : Listening on port 6666 for tcl connections

Info : Listening on port 4444 for telnet connections

Info : clock speed 6000 kHz

**Info : JTAG tap: M2S090.tap tap/device found: 0x0f8031cf (mfg: 0x0e7 (GateField), part: 0xf803, ver: 0x0)**

**Warn : JTAG tap: M2S090.tap UNEXPECTED: 0x0f8031cf (mfg: 0x0e7 (GateField), part: 0xf803, ver: 0x0)**

**Error: JTAG tap: M2S090.tap expected 1 of 1: 0x0f8071cf (mfg: 0x0e7 (GateField), part: 0xf807, ver: 0x0)**

Error: Trying to use configured scan chain anyway...

Warn : Bypassing JTAG setup events due to errors

Info : Listening on port 3333 for gdb connections

Started by GNU MCU Eclipse

Info : accepting 'gdb' connection on tcp/3333

Error: Target not examined yet

undefined debug reason 7 - target needs reset

Error: Target not examined yet

Error: Target not examined yet

Error: Target not examined yet

Possible cause: Using different target while having different target launcher.

Fix: As mentioned in the beginning of the release notes the launchers have to be changed depending on the target. Check exactly what target you have and change it in the debug launcher:

--command "set DEVICE M2S090"

to match the target device

## Application traps at random places or even before reaching main

This can be caused by many causes but frequently repeated cause is when users ignore the instructions of using newer up-to-date HAL. The HAL in the bundled examples was tested and made sure it does behave properly. It was tested with supplied workspaces (workspace.examples and workspace.empty). Use of older or bespoke HAL or even using custom workspace can cause unexpected issues. Follow the Quick start guide sections to avoid any future problems.

Even if the project will import from correctly the code is not migrated. Manual steps are required to migrate a project:
https://github.com/RISCV-on-Microsemi-FPGA/SoftConsole

One change in the HAL covers GP relaxation issue and without current HAL older projects will frequently malfunction:
https://gnu-mcu-eclipse.github.io/arch/riscv/programmer/

## Running SoftConsole from command line

If for some reason the SoftCosole needs to be executed from command line then the appropriate launcher scripts need to be used:

- `softconsole.sh` (Linux OS)
- `softconsole.cmd` (Windows OS)

Running `eclipse.exe` binary directly will not work and cause many issues as the launcher scripts do setup required paths and environment variables for the SoftConsole and Renode to function properly.

## Invalid project path: Duplicate path entries found Warnings

Using macros in include files can cause these warnings, use relative paths on all included folders instead.

RISC-V projects:
*Project settings > C/C++ Build > Settings > Tool Settings > GNU RISC-V Cross C/C++ Compiler > Includes*
For example replace `"${workspace_loc:/${ProjName}/drivers/CoreUARTapb}"` with `"../drivers/CoreUARTapb"`

Arm projects:
*Project settings > C/C++ Build > Settings > Tool Settings > Cross Arm GNU C/C++ Compiler > Includes*
For example replace `"${workspace_loc:/${ProjName}/CMSIS}"` with `"../CMSIS"`

## “No installed packages”

Messages of the following form can be safely ignored. They relate to the GNU MCU Eclipse (CMSIS) Packs support which SoftConsole does not use at this point in time.

```
2018-11-26 12:31:36
Extracting devices & boards...
Loading repos summaries...
Parsing cached content file
"...\extras\Packages\.cache\.content_www_keil_com_pack_index_pidx.xml"...
File does not exist, ignored.
Identifying installed packages...
Found no installed packages.
Completed in 1ms.
No installed packages.
Completed in 1ms.
```

```
2018-11-26 13:55:54
Extracting devices & boards...
No installed packages.
Completed in 1ms.
```

# Other useful Documentation

1. Microsemi github: https://github.com/RISCV-on-Microsemi-FPGA
2. RISC-V specifications: https://riscv.org/specifications/
3. 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.
4. The websites and documentation links for the various open source components used in SoftConsole are also useful references. These are listed elsewhere in this document.

# 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 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



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**.



XXXXXXXX-28Nov2018