Microchip SoftConsole v2021.1

Release Notes

Table of Contents

Microchip SoftConsole v2021.1

Introduction

These are release notes for Microchip SoftConsole v2021.1.

This document uses `<SoftConsole-install-dir>` as a placeholder for the actual SoftConsole install directory. Where this is mentioned substitute the actual SoftConsole install directory name (e.g. `C:\Microchip\SoftConsole_v2021.1` on Windows or `$HOME/Microchip/SoftConsole_v2021.1` on Linux).

Overview

New in v2021.1

- Support for programming PolarFire SoC boot mode 0/1/2/3. Please refer to `<SoftConsole-install-dir>/extras/mpfs/mpfs-bootmodes-readme.txt` for full details.
- Original SoftConsole v6.4 Bash script-based support for boot modes 0/1 replaced by a Java utility program that supports all boot modes.
- Improved custom Eclipse launcher for SoftConsole.
- Fixed a bug in CDT whereby register groups that were previously configured got lost whenever a debug connection attempt failed.
- Please refer to the table in the section below for details of FlashPro programmer support in SoftConsole v2021.1.

New in v6.5

- Support for PolarFire SoC Icicle board's Embedded FlashPro6 Rev B for debugging a PolarFire SoC MSS (Microcontroller Sybsystem) RISC-V multicore core complex based SoC.

  Important note: The Icicle board's Embedded FlashPro6 programmer must be upgraded from Rev A to Rev B for it to work with SoftConsole. Refer to the Libero SoC v12.5 SP1 or Program Debug Tools v12.5 SP1 (or later) release notes for details of how to upgrade the Embedded FlashPro6 from Rev A to Rev B.

  The following table summarizes the FlashPro support in SoftConsole v6.5 and later:

FlashPro → Supports? ↓	FlashPro3/LCPS (standalone & embedded)	FlashPro4 (standalone & embedded)	FlashPro5 (standalone & embedded)	FlashPro6	Embedded FlashPro6 Rev B	Embedded FlashPro6 Rev A
Windows	Y	Y	Y	Y	Y	N
Linux	N	N	Y	Y	Y	N
Virtual Machine	N	N	N	N	N	N
Debug PolarFire SoC MSS Core Complex	Y	Y	Y	Y	Y	N

PolarFire SoC UltraSoC trace/debug	Y	Y	Y	Y	N	N
Debug Mi-V RV32 RISC-V soft core via CoreJTAGDebug/UJTAG	Y	Y	Y	N	N	N
Debug Mi-V RV32 RISC-V soft core via JTAG signals on I/O pins	Y	Y	Y	Y	N	N
Debug Cortex-M1 soft core via CoreJTAGDebug/UJTAG	Y	Y	Y	N	N	N
Cortex-M1 soft core debug via JTAG signals on I/O pins	Y	Y	Y	Y	N	N

New in v6.4

- PolarFire SoC boot mode 0/1 programming support – please refer to <SoftConsole-install-dir>/extras/mpfs/readme.txt for full details.

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, debugging and emulation.
- Support for Microsemi PolarFire SoC, Microsemi SmartFusion® and SmartFusion2 Arm® Cortex-M3, Microsemi Arm® Cortex-M1 and Microsemi Mi-V RISC-V firmware development, debugging and emulation.
- 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 (FlashPro3/4/5/6 on Windows, FlashPro5/6 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 hardware targets allowing development/debugging of software for these targets even when hardware is not available.
- Includes a built-in terminal emulator for connecting to a target board's serial port or the host OS's command shell.
- 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, debug and emulation of bare metal and lightweight RTOS based embedded applications.

Features not supported

- Workspaces, projects and debug launch configurations from some earlier SoftConsole releases are not compatible with SoftConsole v2021.1

    - Workspaces and projects created with SoftConsole releases prior to v4.0 are not compatible and must be recreated.
    - RISC-V projects created with SoftConsole v5.1 or earlier are not compatible and must be recreated.
    - Renode debug launch configurations/launch groups for RISC-V created with SoftConsole v6.0 or earlier are not compatible and must be recreated.
    - In general, it is recommended that any existing workspace, project and debug launch configuration/group be recreated based on the SoftConsole v2021.1 example workspace, projects and debug launch configurations/groups and reviewed to ensure that all configuration settings are appropriate.
- 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.
- SoftConsole does not support or work on any 32-bit Windows or Linux operating system.
- Core8051/Core8051s firmware development and debugging. Use Keil C51 Development Tools with Core8051/Core8051s ISD-51 support.

Quick start guide

1. It is strongly recommended to read the Installation section of Release Notes before the installation as there might be extra manual steps required (especially on Linux).

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.

   Note: Do not launch the `eclipse.exe` binary directly as SoftConsole requires certain environment variables to be set up locally to function properly. Instead run SoftConsole using the provided scripts – `softconsole.cmd` on Windows and `softconsole.sh` on Linux.

5. Do not use custom workspaces or workspaces from previous SoftConsole releases. Use the bundled workspaces, either `workspace.examples` which has bundled examples or `workspace.empty` which is empty has no imported projects. If a new workspace is required then copy and rename `workspace.empty`.

   Sharing/copying workspaces between machines or OSs is inadvisable and may lead to problems.

6. The project settings must match the target hardware (and your Libero design) to use the projects on an actual board.

   If the example project is very different from the target hardware, then generate relevant HAL/CMSIS and firmware drivers from Libero, the Firmware Catalog or the PolarFire SoC Bare Metal Library Bitbucket repository and copy the generated files into the project replacing the existing files which were targeting the original hardware. Make sure that configuration details such as clock speeds are matched in the firmware project and the target hardware.

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. In some cases, these changes can be as minor as changing `set DEVICE M2S090` to `set DEVICE M2S025` in the debug launcher to target the M2S025 devices.

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), the Microsemi GitHub (https://github.com/RISCV-on-Microsemi-FPGA) and/or the PolarFire SoC Bare Metal Library Bitbucket repository (https://bitbucket.microchip.com/projects/FPGA_PFSOC_ES/repos/polarfire-soc-bare-metal-library) to obtain example/demo/reference Libero designs and SoftConsole projects. See also the bundled examples in the `workspace.examples` for simple illustrative examples.

9. It is recommended to read the release notes in full as the document describes the correct way of using SoftConsole, ways to avoid problems and describes the known issues and ways to address them. When troubleshooting an issue, copy the error (or a part of the error) and search for it the Release Notes, the sections contain the error messages to make their solutions easier to find.

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)
      - 7
    - Ubuntu
      - 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

- JTAG Debug
  - Microsemi FlashPro3, FlashPro4, FlashPro5, FlashPro6 and Embedded FlashPro6 Rev B on Windows
  - Microsemi FlashPro5, FlashPro6 and Embedded FlashPro6 Rev B 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 v12.6
    - Microsemi Firmware Catalog v12.6
    - Programming and Debug Tools v12.6
  - Firmware (minimum required version)
    - PolarFire SoC PSE_HAL 1.5.107
    - RISC-V Hardware Abstraction Layer (HAL) 3.0.105
    - SmartFusion2 CMSIS Hardware Abstraction Layer 2.3.105
    - SmartFusion CMSIS-PAL 2.4.102
    - Cortex-M1 CMSIS Hardware Abstraction Layer 2.0.105
    - (DirectCore) Hardware Abstraction Layer 2.3.103

Free/Open source packages

Packages used

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

Java runtime	
Version	OpenJDK 8u252
Home page	https://openjdk.java.net/
Documentation	https://openjdk.java.net/faq/
License	GPL v2 with the Classpath Exception https://openjdk.java.net/legal/
Notes	OpenJDK provides the base Java platform on which Eclipse/CDT and other Eclipse plugins run. Credit/thanks to Oracle and the OpenJDK open source development community.
Eclipse/CDT	
Version	Eclipse 4.15 2020-03 (SimRel) + CDT 9.11
Home page	https://www.eclipse.org/downloads/packages/release/2020-03/r https://projects.eclipse.org/projects/tools.cdt/releases/9.11.0
Documentation	https://help.eclipse.org/2020-03/index.jsp
License	Eclipse Public License v2.0 https://www.eclipse.org/legal/epl-2.0/
Notes	Eclipse/CDT – in conjunction with the GNU MCU Eclipse plugins – provide the main SoftConsole GUI/IDE (Graphical User Interface/Integrated Development Environment). The Windows Eclipse/CDT starter.exe has been modified by Microchip to allow for graceful termination of OpenOCD or other external executables launched from Eclipse. SoftConsole builds from the Eclipse Platform Runtime Binary (PRB) package and only adds those plugins that are absolutely necessary for Microchip target development/debug to avoid unnecessary bloat. Users can still install any additional plugins that they see fit. Credit/thanks to the Eclipse/CDT developer community.
GNU MCU Eclipse Plugins	
Version	v4.7.1-201911052135 (https://gnu-mcu-eclipse.github.io/blog/2019/11/06/plugins-v4.7.1-201911052135-released/)
Home page	https://gnu-mcu-eclipse.github.io/
Documentation	https://gnu-mcu-eclipse.github.io/
License	Eclipse Public License v1.0 https://gnu-mcu-eclipse.github.io/licenses/plug-ins/#eclipse-public-license The copyright owner for all the GNU MCU Eclipse plug-ins is Liviu Ionescu and all rights are reserved.
Notes	Only the following GNU MCU Eclipse plugins are used:

	• 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. • 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. Credit/thanks to Liviu Ionescu and the GNU MCU Eclipse project and development community.
GNU Arm Embedded GCC Toolchain	
Version	xPack GNU Arm Embedded GCC toolchain v9.2.1-1.1-20191025 build based on GNU Arm Embedded Toolchain 9-2019-q4-major (https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads)
Home page	https://xpack.github.io/arm-none-eabi-gcc/
Documentation	https://developer.arm.com/open-source/gnu-toolchain/gnu-rm
License	https://launchpad.net/gcc-arm-embedded/5.0/5-2016-q3-update/+download/license.txt
Notes	Provides GCC, GDB, binutils, newlib (including newlib-nano) etc. development/debug tools for Arm targets (in particular Cortex-M targets). Details of the specific versions of the individual tools in each release package can be found on the Arm Developer website. SoftConsole bundles the xPack GNU Arm Embedded GCC build of the GNU Arm Embedded Toolchain because it offers more cross platform (especially Linux) portability than the Arm build of the same tools and also Windows 64-bit builds (Arm currently provide only 32-bit builds for Windows). Credit/thanks to Arm and the GNU Arm Embedded Toolchain development community and Liviu Ionescu and the xPack and GNU MCU Eclipse projects and development communities.
Arm Cortex Microcontroller Software Interface Standard (CMSIS)	
Version	V4.5
Home page	https://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php
Documentation	http://www.keil.com/pack/doc/CMSIS/General/html/index.html
License	Apache 2.0 License: http://www.keil.com/pack/doc/CMSIS/General/html/index.html#License
Notes	Along with the Microsemi SmartFusion2 CMSIS Hardware Abstraction Layer and SmartFusion CMSIS-PAL firmware packages provides a lightweight hardware abstraction layer on which startup code and firmware drivers can operate. Credit/thanks to Arm.
GNU RISC-V Embedded GCC Toolchain	
Version	xPack GNU RISC-V Embedded GCC toolchain v8.3.0-1.1 build with additional multilibs
Home page	https://xpack.github.io/riscv-none-embed-gcc/
Documentation	https://xpack.github.io/riscv-none-embed-gcc/
License	https://github.com/riscv/riscv-gnu-toolchain/blob/master/LICENSE

Notes	The riscv64-unknown-elf prefixed tools support 32-bit and 64-bit targets and multilibs for all architectures ("native" abi only) excluding any E architectures. Slightly modified versions of the xPack GNU RISC-V Embedded GCC build scripts (https://xpack.github.io/riscv-none-embed-gcc/) are used to build the SoftConsole GNU RISC-V Embedded GCC Toolchain. Multilibs are provided for the following RISC-V arch (architecture) and abi tuples: • march=rv32i/mabi=ilp32 • march=rv32ia/mabi=ilp32 • march=rv32iac/mabi=ilp32 • march=rv32iaf/mabi=ilp32f • march=rv32iafc/mabi=ilp32f • march=rv32ic/mabi=ilp32 • march=rv32if/mabi=ilp32f • march=rv32ifc/mabi=ilp32f • march=rv32im/mabi=ilp32 • march=rv32ima/mabi=ilp32 • march=rv32imac/mabi=ilp32 • march=rv32imaf/mabi=ilp32f • march=rv32imafc/mabi=ilp32f • march=rv32imc/mabi=ilp32 • march=rv32imf/mabi=ilp32f • march=rv32imfc/mabi=ilp32f • march=rv64i/mabi=lp64 • march=rv64ia/mabi=lp64 • march=rv64iac/mabi=lp64 • march=rv64iaf/mabi=lp64f • march=rv64iafc/mabi=lp64f • march=rv64iafd/mabi=lp64d • march=rv64iafdc/mabi=lp64d • march=rv64ic/mabi=lp64 • march=rv64if/mabi=lp64f • march=rv64ifc/mabi=lp64f • march=rv64ifd/mabi=lp64d • march=rv64ifdc/mabi=lp64d • march=rv64im/mabi=lp64 • march=rv64ima/mabi=lp64 • march=rv64imac/mabi=lp64 • march=rv64imaf/mabi=lp64f • march=rv64imafc/mabi=lp64f • march=rv64imafd/mabi=lp64d • march=rv64imafdc/mabi=lp64d • march=rv64imc/mabi=lp64 • march=rv64imf/mabi=lp64f • march=rv64imfc/mabi=lp64f • march=rv64imfd/mabi=lp64d • march=rv64imfdc/mabi=lp64d This is a full list of supported multilibs. Arch/abi tuples not listed are not supported. This includes all RV32E targets. Note that this suite of multilibs supports the "natural" ABI for each architecture. If an RV64G (rv64imafd or more specifically rv64imafdZicsr_Zifencei) architecture is chosen, but not the natural ABI lp64f (natural would be in this case lp64d), then the resulting multilib is not supported. However, choosing the rv64imaf with its natural ABI lp64f will work. Instead of selecting architecture and then ABI a reversed approach might be

	better. Select an ABI to fulfil application's requirements and then the select the minimal architecture that can handle that given ABI. The "default/native" mode of the tools is RV64GC in the absence of any -march/-mabi flags being passed to the tools. Credit/thanks to the RISC-V development community and Liviu Ionescu and the xPack and GNU MCU Eclipse projects and development communities.
OpenOCD	
Version	v0.10.0+dev based on the SiFive fork of OpenOCD with additional SoftConsole and UltraSoC mods
Home page	http://openocd.org/
Documentation	http://openocd.org/documentation/
License	GNU General Public License v2 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 using real hardware. When debugging Eclipse launches GDB and then uses GDB's GDB/MI (Machine Interface) to communicate with the debugger. Meanwhile GDB communicates with OpenOCD using OpenOCD's Remote Serial Protocol interface. GDB debug operations are translated into JTAG operations by OpenOCD which communicates with the target hardware/CPU using JTAG via the relevant JTAG debug probe. OpenOCD also has knowledge of specific CPU target debug frameworks (e.g. Arm CoreSight, RISC-V Debug Modules v0.11 and v0.13 – https://github.com/riscv/riscv-debug-spec) so that it can communicate with and debug supported RISC-V CPUs. SoftConsole OpenOCD is based on the SiFive fork of OpenOCD (https://github.com/riscv/riscv-openocd) with additional SoftConsole modifications to add support for FlashPro, SmartFusion2/SmartFusion/Fusion eNVM, finding scripts relative to OpenOCD bin directory, other fixes and enhancements. It is also built using some UltraSoC mods to support UltraDevelop trace functionality. SoftConsole's version of OpenOCD is built using slightly modified versions of the xPack OpenOCD (https://xpack.github.io/openocd/) build scripts.. Credit/thanks to the OpenOCD development community, SiFive, UltraSoC and Liviu Ionescu and the xPack and GNU MCU Eclipse projects and development communities.
GNU MCU Eclipse Build Tools	
Version	v2.12-20190422 https://gnu-mcu-eclipse.github.io/windows-build-tools/releases/
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/ make: GNU General Public License v3 http://www.gnu.org/copyleft/gpl.html BusyBox GNU General Public License v2 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 and busybox for Windows and Linux. Credit/thanks to Liviu Ionescu and the GNU MCU Eclipse project and development community.
Antmicro Renode	
Version	v1.11.0.8104 (fc14a2fe-202102120426)
Home page	https://github.com/renode https://renode.io/
Documentation	https://renode.readthedocs.io/en/latest/
License	MIT License: https://github.com/renode/renode/blob/master/LICENSE
Notes	Renode can emulate physical hardware systems - including the CPU, peripherals, sensors, environment, and wireless medium between nodes. Credit/thanks to Antmicro and the Renode development community.
Cppcheck	
Version	v1.83
Home page	http://cppcheck.sourceforge.net/
Documentation	https://sourceforge.net/p/cppcheck/wiki/Home/
License	GNU Public License v3 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. Credit/thanks to Daniel Marjamäki and the cppcheck development community.
Cppcheclipse	
Version	v1.1.1-20180105 – build based from commit 7bae5b2
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. Credit/thanks to the Cppcheck development community.
Python	
Version	v3.8.2
Home page	https://en.wikipedia.org/wiki/InstallJammer
Documentation	http://installjammer.com/docs/ (domain expired) https://sourceforge.net/p/installjammer/wiki/Home/
License	Python Software Foundation License https://docs.python.org/3/license.html
Notes	Credit/thanks to the Python development community.

UltraSoC UltraDevelop	
Version	v2.0.1
Home page	https://www.ultrasoc.com/technology-2/risc-v/
Documentation	https://www.ultrasoc.com/technology-2/risc-v/
License	Commercial
Notes	UltraSoC UltraDevelop components supporting trace/debug access to PolarFire SoC. Credit/thanks for UltraSoC.
VMware InstallBuilder	
Version	v20.12.0
Home page	https://installbuilder.com/
Documentation	https://clients.bitrock.com/installbuilder/docs/installbuilder-userguide/index.html
License	Commercial: https://installbuilder.com/purchase.html VMware/BitRock have kindly provided an open source license (https://support.bitrock.com/hc/en-us/community/posts/115002024525) allowing InstallBuilder to be used to create the SoftConsole installers for Windows and Linux.
Notes	Credit/thanks to VMware (formerly BitRock).

Installation

Notes

SoftConsole does not yet support installations shared between multiple users so for the moment a separate instance must be installed for each user.

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 `Microchip-SoftConsole-v2021.1-windows-x64-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.

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

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

In a rare case when the .NET Framework is not already installed with Windows 8.1/10 then it can be easily added by clicking on the Windows *Start* button, typing *Turn Windows features on or off* and then selecting the relevant *.NET Framework* to be installed/enabled from the *Windows Features* dialog then click *OK*.

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 for Linux is now fully 64-bit and unlike previous releases no longer requires additional 32-bit packages installed on Linux 64-bit.

1. The installer is a 64-bit executable GUI based program named `Microchip-SoftConsole-v2021.1-linux-x64-installer.run` (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 Microchip-SoftConsole-v2021.1-linux-x64-installer.run
```

3. Run the installer:

```
./Microchip-SoftConsole-v2021.1-linux-x64-installer.run
```

4. Some desktop/window managers might not put the installer into the focus or show it in the taskbar. The installation might seem frozen, try to minimalize the terminal window to see the installation dialog underneath.

5. Follow the installer GUI wizard or console mode instructions on screen.

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

Post installation steps

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.x and 6.x) may not work and are not recommended or supported.

Post installation steps for Renode emulation package

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

1. Visit **http://www.mono-project.com/download/stable/**, follow the instructions for your distribution and add the relevant Mono repository to your system (do not install packages yet).
    a. Note: The Mono Project website assumes that `sudo` is installed and configured for regular users. On restricted or minimalistic systems the commands might need to be adjusted slightly and run directly using `su` or the root account itself.
2. Install `mono-complete` instead of `mono-develop` (see the steps below)
    a. Note: Renode 1.9.0 requires Mono v5.20 or higher.
3. Some distributions need GTK related packages (`gtk-sharp2` and `libcanberra-gtk` are installed below).

After the mono repository has been added, the following commands need to be run:

Debian/Ubuntu:

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

CentOS/Red Hat Enterprise Linux:

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

openSUSE:

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

Note:

OpenSUSE Leap 15 is not supported by the Mono project and the packages supplied with the Linux distribution must be used. Leap 15 includes Mono v5.10 which is insufficient for Renode.

Post installation steps for hardware targets (FlashPro5)

By default, USB devices are only accessible with root privileges. To debug using SoftConsole and FlashPro5/6 as a non-root user some additional steps must be taken.

1. Copy the OpenOCD udev rules file and tell the udev subsystem 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
   ```

   If `/etc/udev/rules.d/60-openocd.rules` already exists then make sure to overwrite it with the new version in this release as it contains additional rule data not in the earlier version.

   In some cases it may be necessary to reboot for the changes to take effect. Some distributions do not create the `plugdev` group and/or may not add users to it automatically. In that case the `plugdev` group must be created manually and the user added to the `plugdev` group.

   ```
   sudo groupadd plugdev
   sudo usermod -a -G plugdev <username>
   ```

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 superseded by `60-openocd.rules`. Then run `udevadm trigger` again or reboot for the changes to take effect.

3. To check that FlashPro5/6 can be used without root privileges…

   Connect a FlashPro5/6 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
   Bus 001 Device 004: ID 1514:2009 Actel
   ```

   If the FlashPro5 (VID=1514 PID=2008) or FlashPro6 (VID=1514 PID=2009) does not appear then double check that the previous steps were carried out correctly.

4. To the JTAG end of the FlashPro 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
   ```

   ```
   ./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
   ```

OR

```
./openocd -c "set DEVICE MPFS" -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
```

Related Microsemi Tools/Resources

Libero SoC/Firmware Catalog

Use Microsemi Libero SoC v12.6 or later to create hardware designs and to export firmware drivers and example projects.

The Microsemi Firmware Catalog can be used to generate firmware drivers and example projects for use in SoftConsole v2021.1.

Firmware drivers

Hardware Abstraction Layers

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

- PolarFire SoC MPFS_HAL v1.5.107
- RISC-V Hardware Abstraction Layer (HAL) 3.0.105
- SmartFusion2 CMSIS Hardware Abstraction Layer 2.3.105
- SmartFusion CMSIS-PAL 2.4.102
- Cortex-M1 CMSIS Hardware Abstraction Layer 2.1.101
- (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.

The table below summarizes which FlashPro variants are supported on which OS by SoftConsole.

FlashPro ↓ / OS Support? →	**Linux**	**Windows**
FlashPro3	N	Y
FlashPro4	N	Y
FlashPro5	Y	Y
FlashPro6 (standalone)	Y	Y
FlashPro6 (embedded – Icicle board)	N	N

On Windows the FlashPro3/4/5/6 programmers are supported, and the relevant drivers must be installed. On Linux, the FlashPro5/6 programmers are supported and the post-install configuration steps must be carried out to allow access to the programmer by non-root users.

Microsemi’s GitHub repositories

The Microsemi GitHub (https://github.com/RISCV-on-Microsemi-FPGA) is a useful Microsemi Mi-V ecosystem resource. However, the bundled examples with SoftConsole v2021.1 are tested and are working with SoftConsole v2021.1 correctly, therefore are recommended as the primary reference.

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. Workspaces were not designed nor intended to be shared between machines nor operating systems. Do not make your own empty workspaces and do not transfer workspaces between machines or OSes.

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

- mpfs-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 and Renode sections for more information.
- mpfs-mustein-julia: A Renode-only example showcasing a simple graphical peripheral and own platform
- mpfs-freertos-lwip: Demonstration of a FreeRTOS implementation running a LwIP and a Webserver.
- 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`

This workspace can be used if the bundled example projects are not desired, but still contains the Renode platform launchers and all other necessary settings. This workspace is ideal for importing user’s projects into.

Creating a new workspace

To create a new workspace it is recommended to make a copy of the `workspace.empty` example workspace and open that. The `workspace.empty` (and workspace.examples) have various useful settings – including an integrated “develop and debug” perspective – preconfigured to make development and debug easier than with a completely new/blank workspace. Bundled Renode platform launchers, various small settings which have a significant impact in some edge cases, custom user dictionary and much more.

All documentation assumes that only the bundled (or cloned) workspaces are used. Because the workspaces change between the releases it’s not supported to re-use older workspace from previous SoftConsole.

Not following this rule and creating own ad-hoc workspaces might on the surface seem as working but it will eventually cause various problems. These issues might not seem to be relevant to the workspace and might be non-trivial to resolve.

Copying workspaces between hosts and especially different OSs is not supported and can cause various and hard to debug problems as well.

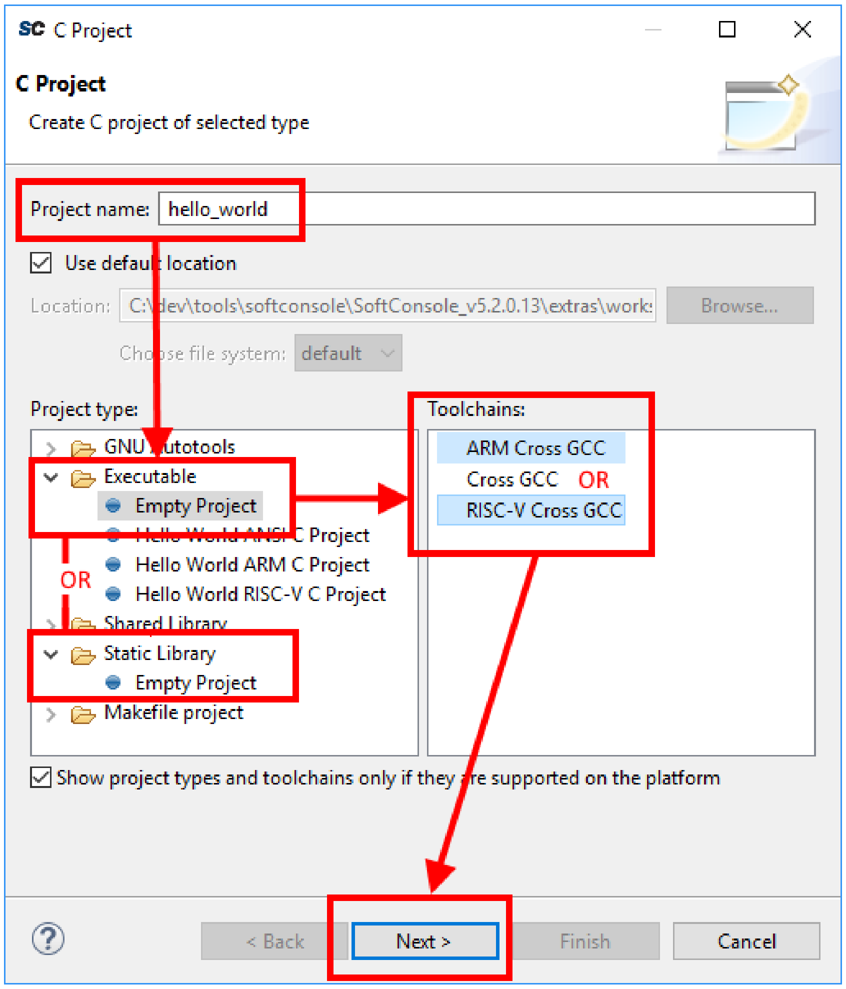
Only SoftConsole v2021.1 workspace.examples, workspace.empty and its copies are supported.

Projects

When making own projects the existing bundled projects can be used as a base reference. For small, quick experiments and proofs-of-concept the existing projects can be used as a donor and the HAL, drivers, and application changed. Preferably make a backup of the project or the whole workspace before changing the existing examples.

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*.


C Project

C Project

Create C project of selected type

Project name: hello_world

☑ Use default location

Location: C:\dev\tools\softconsole\SoftConsole_v5.2.0.13\extras\work:

Choose file system: default

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

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 trap exceptions (mcause = 4 or 6) 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 and RTG4/SmartFusion2/IGLOO2/PolarFire RISC-V targets

For demonstration a SmartFusion2 was chosen, however similar steps can be applied to all other targets and other types of memories.

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 (typing the command is safer as sometime Copy/Paste might copy wrong Unicode characters):

```
--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.

Previously some users deleted the first line of the HEX file, even it might seem like a working solution, but it's flawed on multiple levels and is **strongly not recommended nor supported**. There is a high risk of the data getting malformed and causing various other issues in the deployment. Making new launcher "Deploy" and adding this as the permanent flag will allow users to re-generate the file as frequently as they need without a need of any extra manual steps (and not modify the data beyond the tool's back). And because the Intel HEX format is based on 16-bit blocks, the bigger applications will **not work at all** with the 'delete the first line' workaround (especially the error around the line 4097). This might falsely indicate that there is something wrong with the application, code or the compilation settings/features or something wrong between Debug/Release configurations because with different code/settings/configuration the issue is not observed, however it still might be caused by not "normalizing" the file properly.

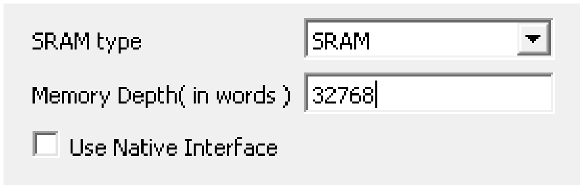
The error messages differ depending on what core/memory the hex file is getting stored into. For example, the error from PF_SRAM_AHBL_AXI module might be different than error from PF_uPROM or eNVM module, however all might be pointing to the same issue. Here is a list (not extensive) of some error messages which might be related to this issue:

Error: Invalid Memory File Content: Errors encountered while parsing the Memory file.

IHX006: Intel Hex Address record error. Address extension specified in the record is more than the maximum possible address for the destination, when reading file "<YOUR_HEX>" at line 1.

IHX006: Intel Hex Address record error. Address extension specified in the record is more than the maximum possible address for the destination, when reading file "<YOUR_HEX>" at line 4097.

There might be other independent but overlapping issues which make it harder for the users to debug this issue. Very common and easy to make mistake is having the physical memory smaller than the application requirements, make sure the memory is sized properly. The meaning of "word" in Libero design might not mean what users could expect and might not correlate with the Data Width settings, users should verify its meaning with the core's documentation. It's possible to set the SRAM block to 64KiB size (word meaning 16-bits no matter what is the Data Bus Width) while users expecting it to be 128KiB size:


SRAM type: SRAM
Memory Depth(in words): 32768
☐ Use Native Interface

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

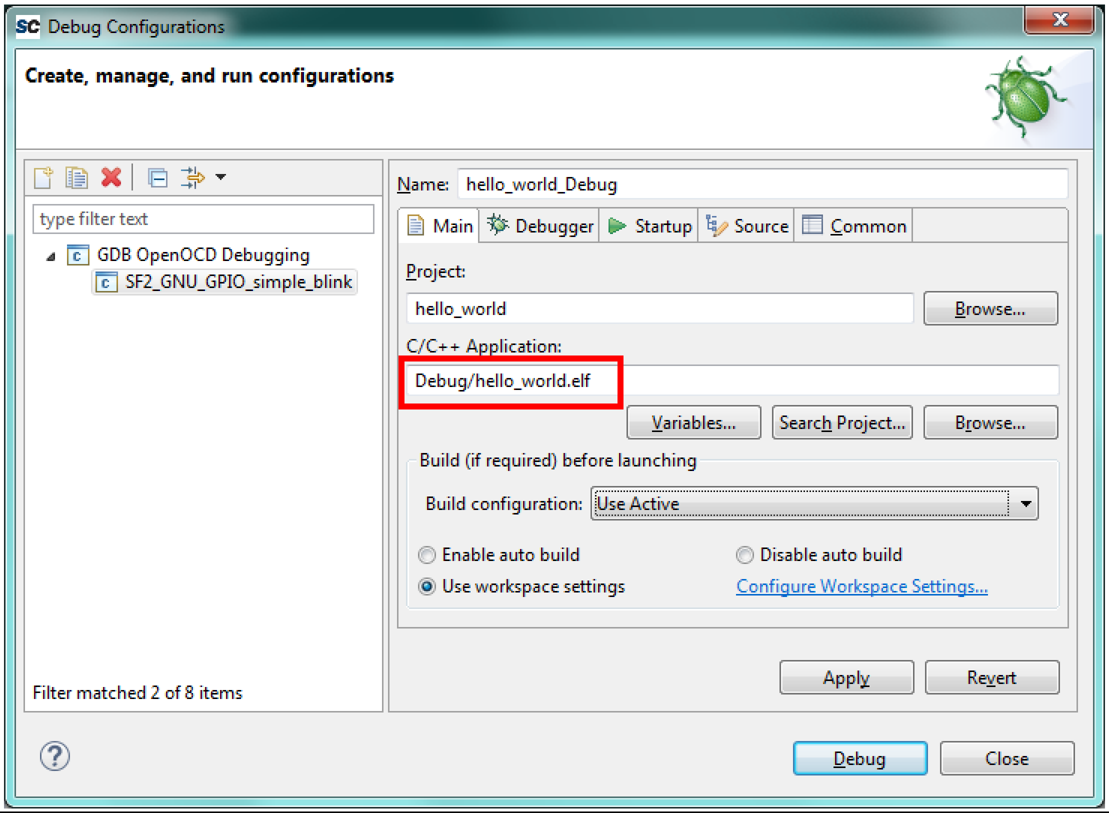
To debug a program a debug launch configuration must be created. Most of the default settings for a debug launch configuration can be left as they are but a few needs to be manually configured. Use the example projects and debug launch configurations as a guide to creating new debug launch configurations.

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:



Name: hello_world_Debug

Project: hello_world

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

Build (if required) before launching

Build configuration: Use Active

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

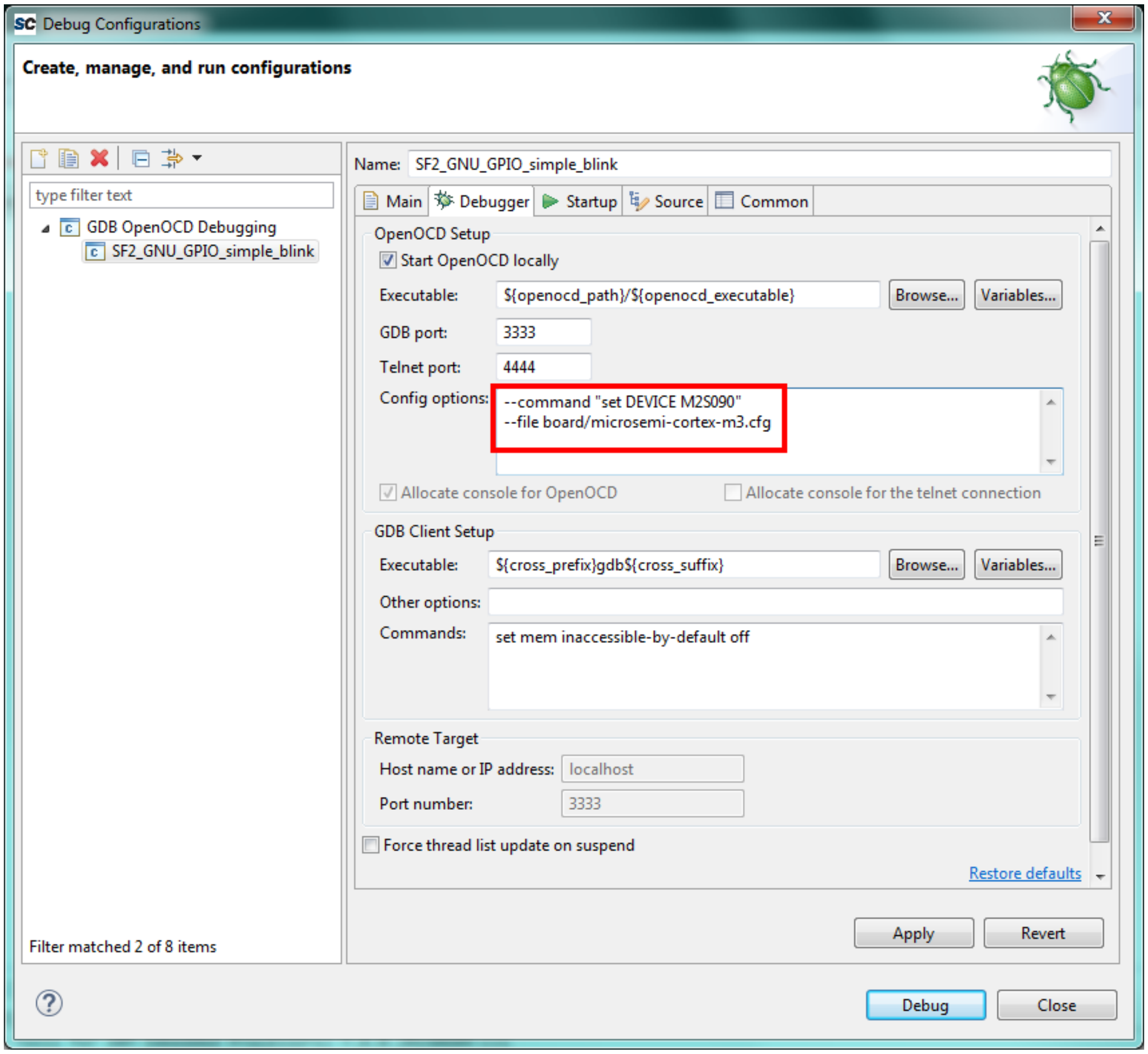
Figure 2. Debug launch configuration Main tab

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
   ```


Debug Configurations

Create, manage, and run configurations

type filter text

GDB OpenOCD Debugging
SF2_GNU_GPIO_simple_blink

Filter matched 2 of 8 items

Name: SF2_GNU_GPIO_simple_blink

Main | Debugger | Startup | Source | Common

OpenOCD Setup

[✓] Start OpenOCD locally

Executable: ${openocd_path}/${openocd_executable}

GDB port: 3333

Telnet port: 4444

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

[✓] Allocate console for OpenOCD   [ ] Allocate console for the telnet connection

GDB Client Setup

Executable: ${cross_prefix}gdb${cross_suffix}

Other options:

Commands: set mem inaccessible-by-default off

Remote Target

Host name or IP address: localhost

Port number: 3333

[ ] Force thread list update on suspend

Restore defaults

Apply Revert

Debug Close

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
   ```

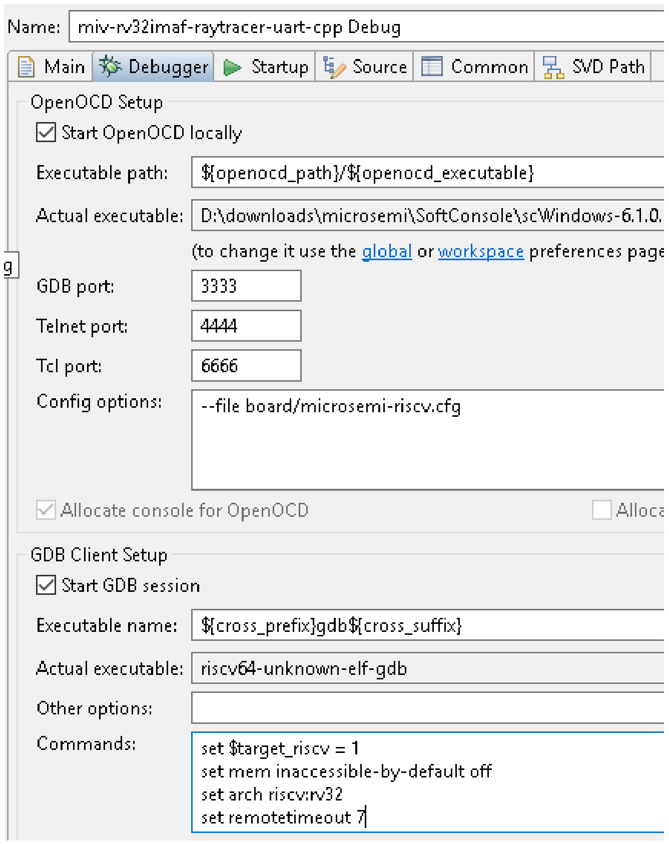
   GDB Client Setup > Commands:
   ```
   set mem inaccessible-by-default off
   set $target_riscv = 1
   ```
   This is necessary even if there are some cases where it seems to work correctly (and previous examples did not require this command)!

   Debugging RISC-V 32-bit targets

   Additionally, when debugging a program on the Renode RISC-V 32-bit emulation model then the bit-size of the target must be specified as well (do not remove the previous commands such as set $target_riscv = 1):
   ```
   set arch riscv:rv32
   ```
   Note: This is not needed when targeting 64-bit PolarFire SoC emulation model.

   When the binary elf file is large then it might sometimes on 32-bit targets cause a timeout message, to suppress these the timeout can be changed:
   ```
   set remotetimeout 7
   ```

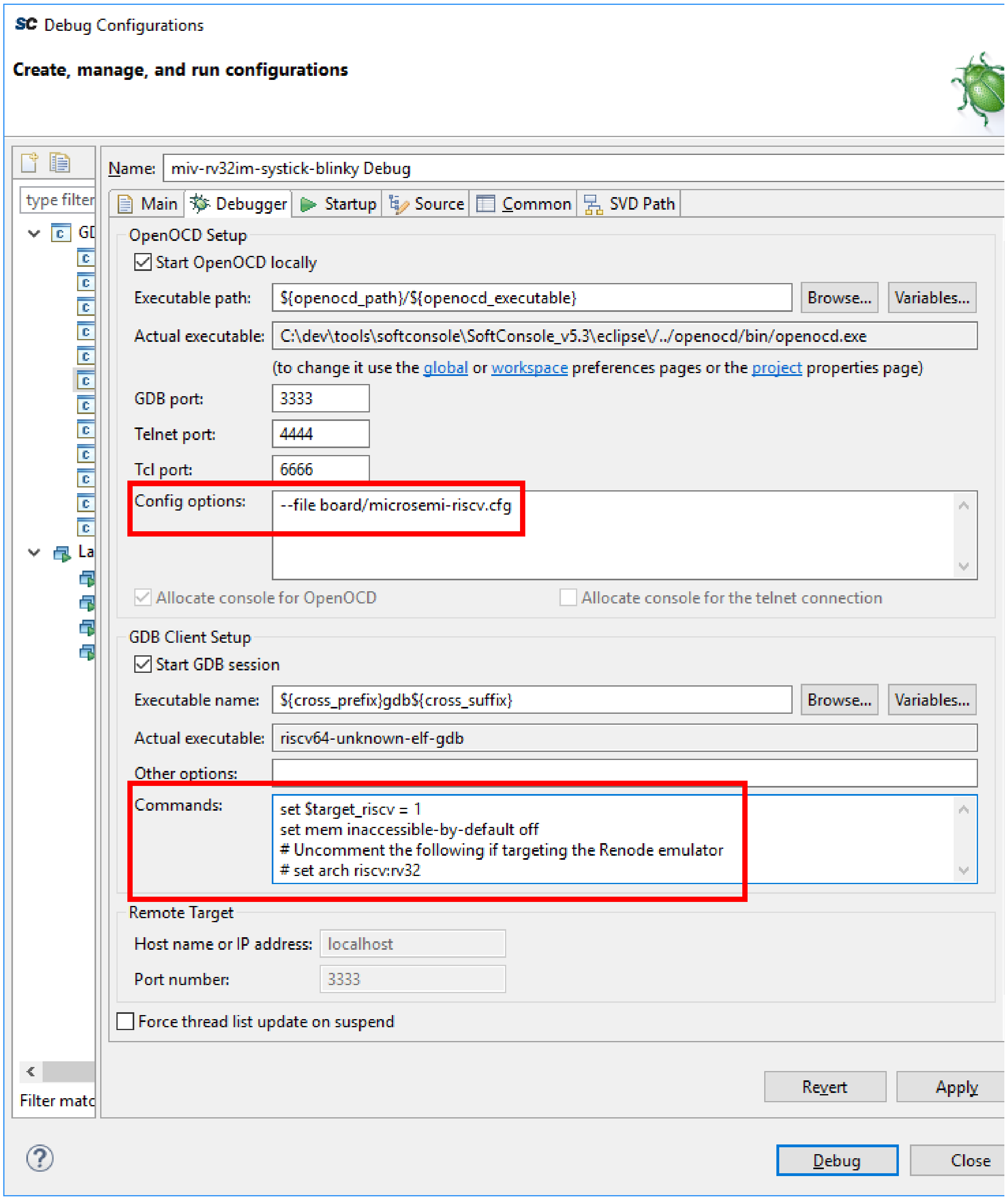
   Existing workspace examples should be used to see how these are configured:



Name: miv-rv32imaf-raytracer-uart-cpp Debug

OpenOCD Setup
☑ Start OpenOCD locally
Executable path: ${openocd_path}/${openocd_executable}
Actual executable: D:\downloads\microsemi\SoftConsole\scWindows-6.1.0.
(to change it use the global or workspace preferences page
GDB port: 3333
Telnet port: 4444
Tcl port: 6666
Config options: --file board/microsemi-riscv.cfg
☑ Allocate console for OpenOCD ☐ Alloca

GDB Client Setup
☑ Start GDB session
Executable name: ${cross_prefix}gdb${cross_suffix}
Actual executable: riscv64-unknown-elf-gdb
Other options:
Commands:
```
set $target_riscv = 1
set mem inaccessible-by-default off
set arch riscv:rv32
set remotetimeout 7
```


Debug Configurations

Create, manage, and run configurations

Name: miv-rv32im-systick-blinky Debug

OpenOCD Setup

☑ Start OpenOCD locally

Executable path: ${openocd_path}/${openocd_executable}

Actual executable: C:\dev\tools\softconsole\SoftConsole_v5.3\eclipse\/../openocd/bin/openocd.exe

(to change it use the global or workspace preferences pages or the project properties page)

GDB port: 3333

Telnet port: 4444

Tcl port: 6666

Config options: --file board/microsemi-riscv.cfg

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

GDB Client Setup

☑ Start GDB session

Executable name: ${cross_prefix}gdb${cross_suffix}

Actual executable: riscv64-unknown-elf-gdb

Other options:

Commands:

```
set $target_riscv = 1
set mem inaccessible-by-default off
# Uncomment the following if targeting the Renode emulator
# set arch riscv:rv32
```

Remote Target

Host name or IP address: localhost

Port number: 3333

☐ Force thread list update on suspend

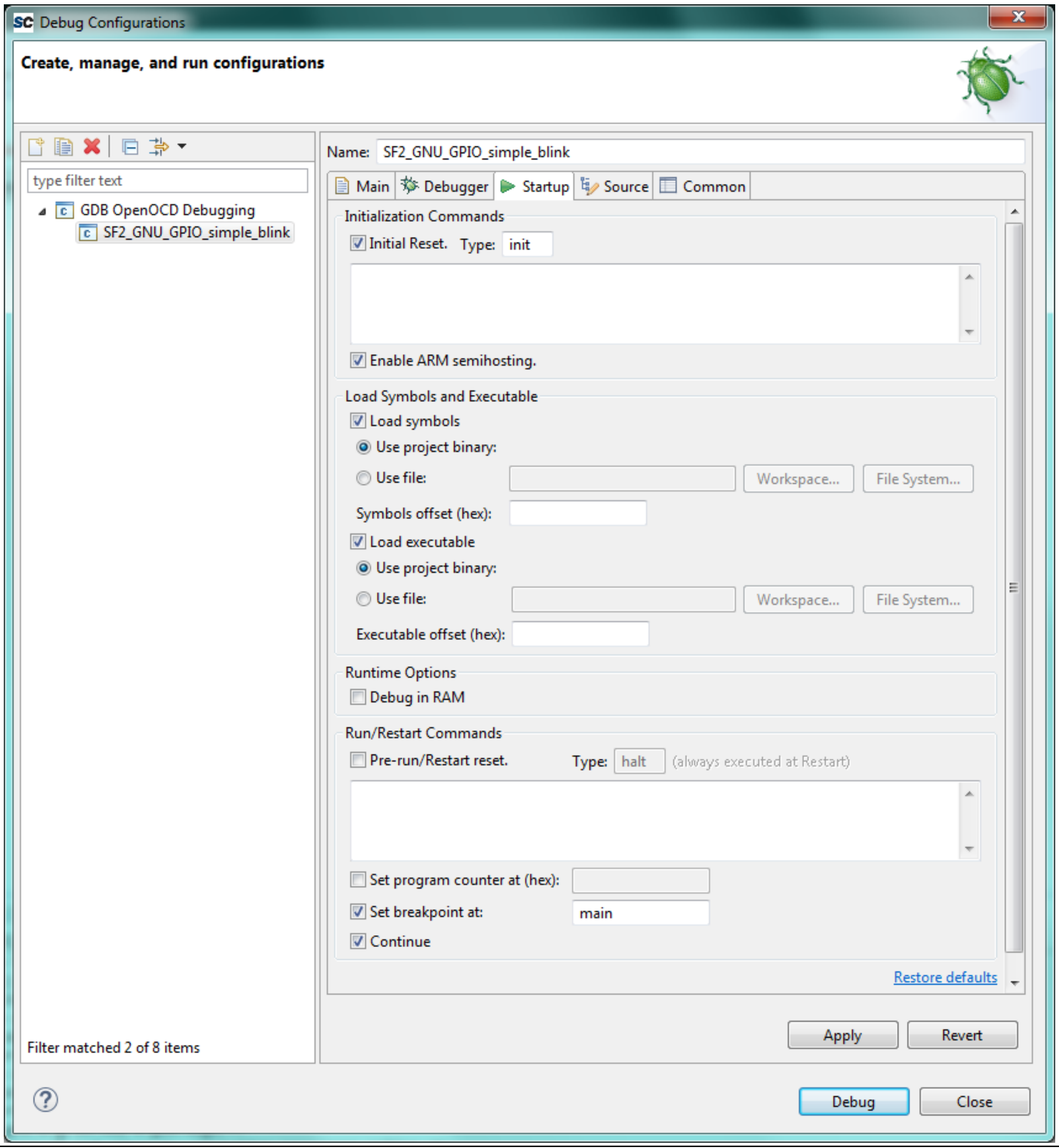
Figure 4. Debug Configuration Debugger tab for RISC-V

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.


Debug Configurations

Create, manage, and run configurations

type filter text

GDB OpenOCD Debugging
SF2_GNU_GPIO_simple_blink

Filter matched 2 of 8 items

Name: SF2_GNU_GPIO_simple_blink

Main | Debugger | Startup | Source | Common

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

Load Symbols and Executable
- [✓] Load symbols
- (●) Use project binary:
- ( ) Use file: Workspace... File System...
- Symbols offset (hex):
- [✓] Load executable
- (●) Use project binary:
- ( ) Use file: Workspace... File System...
- Executable offset (hex):

Runtime Options
- [ ] Debug in RAM

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

Restore defaults

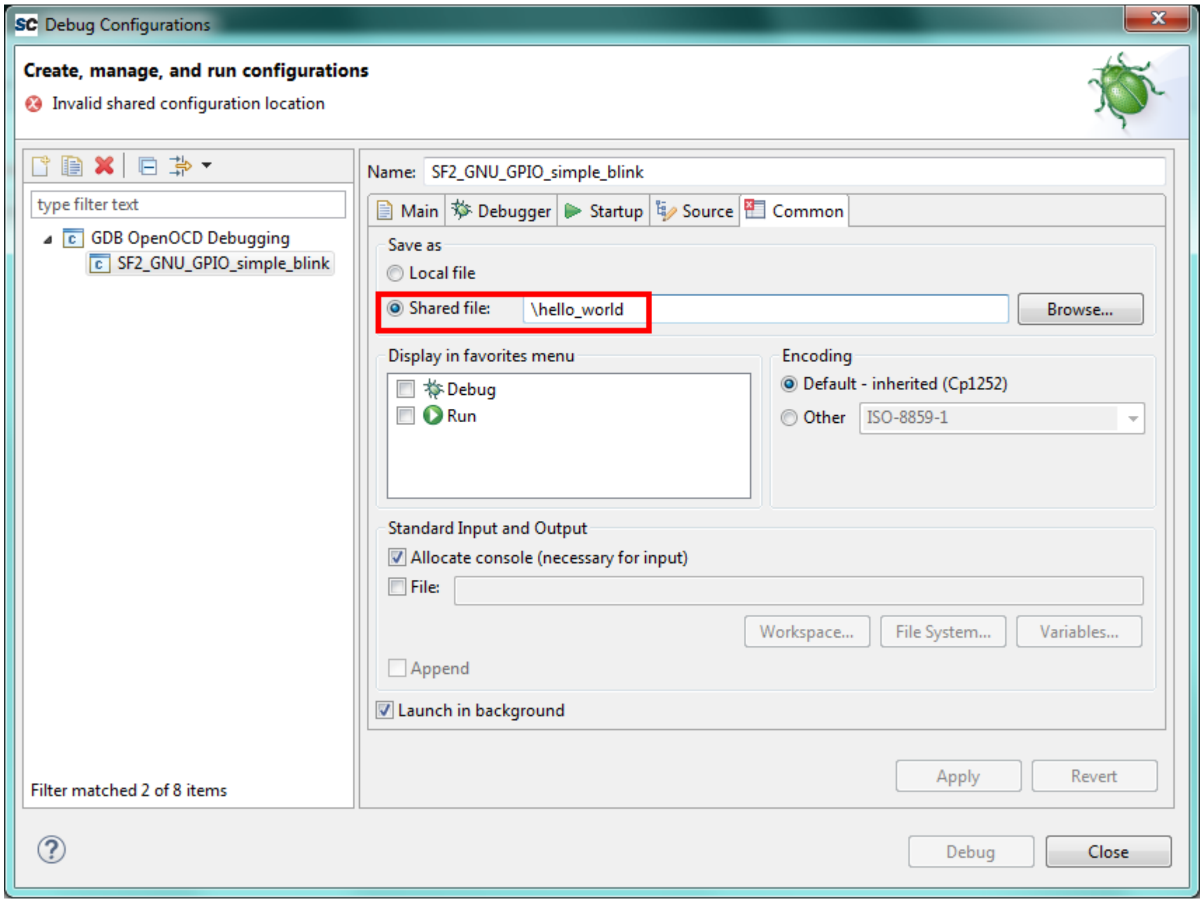
Apply | Revert

Debug | Close

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).

`


Debug Configurations

Create, manage, and run configurations

Invalid shared configuration location

Name: SF2_GNU_GPIO_simple_blink

GDB OpenOCD Debugging
SF2_GNU_GPIO_simple_blink

Filter matched 2 of 8 items

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

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

Allocate console (necessary for input) ✓

Launch in background ✓

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 MPFS_HAL (previously known as PSE_HAL)	
Linker script	**OpenOCD board script**
`Refer to the Mi-V RISC-V HAL, the PolarFire SoC MPFS HAL and the various RISC-V example projects provided for details of the example linker scripts provided.`	`board/microsemi-riscv.cfg` `board/Microsemi-sifive-hifive-unleashed.cfg`
SmartFusion2 CMSIS Hardware Abstraction Layer	
Linker script	**OpenOCD board script**
`debug-in-microsemi-smartfusion2-esram.ld` `debug-in-microsemi-smartfusion2-envm.ld`	`board/microsemi-cortex-m3.cfg`

`debug-in-microsemi-smartfusion2-external-ram.ld`	`board/microsemi-smartfusion2-eval-or-starter-kit-ddr.cfg`
	`board/microsemi-smartfusion2-dev-kit-ddr.cfg`
	`board/microsemi-smartfusion2-dev-kit-ddr-ecc.cfg`

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

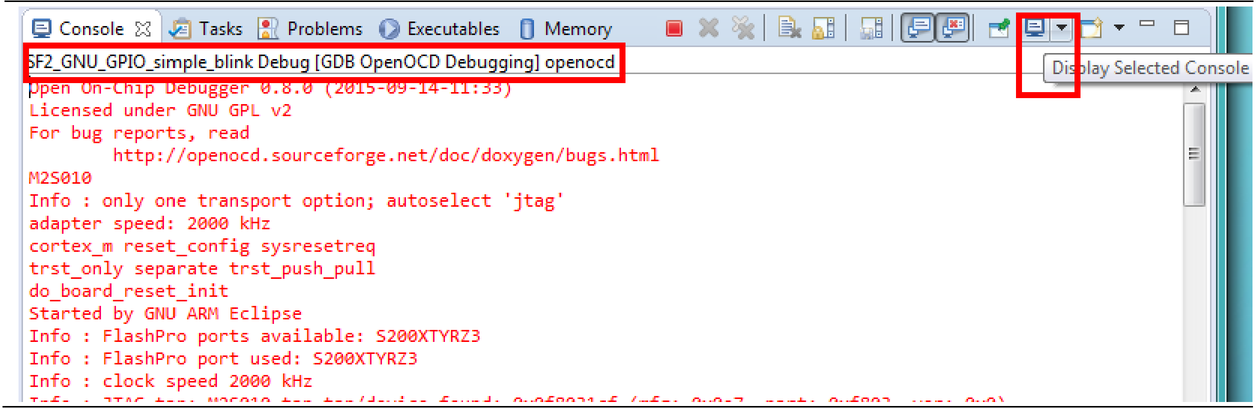
Select the project in the *Project Explorer*, right click on it and from the context menu select *Debug As* > *Debug Configurations*, select the relevant debug launch configuration and click *Debug*.

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

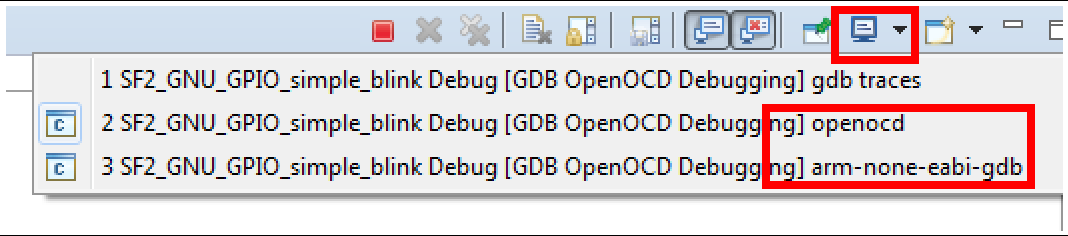
During a debug session SoftConsole can display several different consoles in the *Console* view. By default, the OpenOCD console is displayed showing OpenOCD output:



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

Figure 7. Debug session – OpenOCD console view

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



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

Figure 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 interface/ftdi/olimex-arm-usb-tiny-h.cfg
   --file board/microsemi-cortex-m3.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 the Mi-V model (external tool launcher called *Mi-V-Renode-emulation-platform*) or PolarFire SoC MPFS (previously known as PSE, see MPFS HAL manual for more details) 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 hardware targets except:

1. *Debugger* > *OpenOCD Setup* > *Start OpenOCD locally* must 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`

3. For PolarFire SoC targets the launcher must be changed (this changed between SoftConsole 6.0 and 6.1).

   a. The Startup -> Run/Restart Commands needs to contain the following code:
   ```
   monitor $COMMON_PC=`sysbus.e51 PC`
   monitor runMacro $SetAllPCs
   monitor start
   ```

   b. Change the e51 to u54_1 in the code above if the connection is made to u54_1, change accordingly for u54_2, u54_3, and u54_4.

   c. The SoftConsole 6.2 platform script needs to be used (it's not compatible with 6.2). The SoftConsole 6.2 scripts now start the servers with the auto-start feature disabled (no "true" appended on each line)
   ```
   e51 StartGdbServer 3333
   u54_1 StartGdbServer 3334
   u54_2 StartGdbServer 3335
   u54_3 StartGdbServer 3336
   u54_4 StartGdbServer 3337
   ```

   d. Connect to port with the correct hart, check the Debugger -> Remote Target -> Port number value (e51 = 3333, u54_1 = 3334, u54_2 = 3335, u54_3 = 3336, u54_4 = 3337)

   e. The very first launcher will configure the platform, PCs and start execution of the code. If a second connection is required to debug another hart, then extra steps need to be followed

      i. Base the second launcher by making a duplicate of the first launcher

      ii. In the Debugger tab:

         1. Leave the `Start OpenOCD locally` unchecked

         2. Select a new port to connect to (3333 to 3337 depending what hart is targeted)

      iii. In the Startup tab:

         1. Uncheck `Initial Reset`

         2. Check `Load symbols`

3. Uncheck `Load executable` (loading ELF file again would have an undesired effect)
4. Uncheck `Set breakpoint at`
5. Check `Continue`
6. Remove unneeded `Initial Reset` and `Run/Restart commands` monitor commands. Such as opening the UART monitor, setting the `loglevel` and all commands which will be executed by the first hart. Do **NOT** set the PCs as they were set when the first launcher connected (do not run the `monitor runMacro $SetAllPCs` again). And do not let the emulation start again (`monitor start`) as it is started again.

iv. Add any new additional commands if needed, for example opening second UART monitor for the second hart, increasing `logLevel` of a peripheral which is used by the second hart and it needs to be investigated etc.

f. See the example mpfs-blinky Renode all-harts Start-platform-and-debug launcher which is used to configure everything and start debugging session, while the mpfs-blinky Renode all-harts Attach-to-running is used as a second connection and connects to running emulation.

g. The debug launcher might be tweaked, and unused harts can be forced to permanent sleep. This could improve the emulation performance as they are not emulating infinite loops, but this needs to be supported by HAL. Older HAL from SoftConsole 6.0 was not expecting any harts and (and depends if HAL expects all 5 harts to be present).

If some harts are not utilized, they can be halted from Renode by adding command:
`monitor sysbus.u54_4 IsHalted true`
to the `Debug Launcher -> Startup -> Initialization Commands` section of the desired debug launcher.

Depending on the HAL's version there might be required extra step. On current HAL v1.5 it is necessary to add pre-processor predefine `MPFS_HAL_LAST_HART=<LAST_HART_WHICH_RUNNING>` into the `C/C++ Build -> Settings -> Tool Settings -> GNU RISC-V Cross C Compiler -> Preprocessor` (it might be necessary to be added to all other build configurations). If only last hart (4) is halted then the preprocessor needs to have define `MPFS_HAL_LAST_HART=3`

Setting it to `MPFS_HAL_LAST_HART=0` will make everything after the `MPFS_HAL_LAST_HART` (harts 1-4) as not present in the system.

Previous HAL from SoftConsole v6.0 doesn't have such behavior and extra attention is required when porting projects from SoftConsole v6.0 to v2021.1 and when updating HAL. Either the launchers shouldn't halt any harts, or the predefine needs to be added. A significant difference between the HALs is that the newer HAL has refined and standardized project structure which is not compatible with the older layout. Note: This is not an extensive list of the HAL changes and HAL's manual should be reviewed.

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

- miv-rv32im-interrupt-blinky Renode Start-platform-and-debug
- miv-rv32im-systick-blinky Renode Start-platform-and-debug
- miv-rv32imaf-mandelbrot-uart Renode Start-platform-and-debug
- miv-rv32imaf-raytracer-uart-cpp Renode Start-platform-and-debug
- mpfs-blinky Renode all-harts Start-platform-and-debug
- mpfs-freertos-lwip Renode all-harts Start-platform-and-debug
- mpfs-mustein-julia Renode all-harts Start-platform-and-debug

Compared to SoftConsole 6.0 the launchers have new naming scheme: <project> <target> <hart> <action>

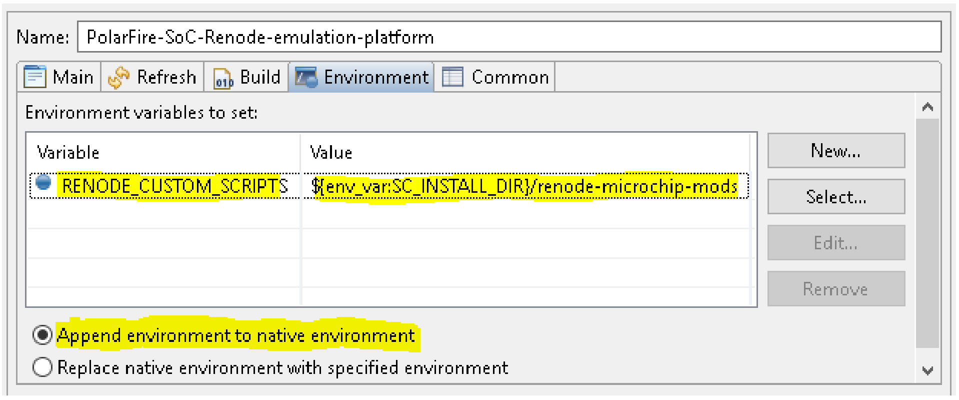
With current launchers, only one instance of Renode is supported, which means that the previous session needs to be completely closed before running a new session. This can be achieved either by selecting the group launcher and in the *Debug* windows and terminating it with red stop icon (Ctrl+F2) in 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 in failing to communicate on the port 3333 (or 3333-3337 as *mpfs-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. If there was no previous error in the launch procedure then in some cases the F11 can be used to re-launch the last launcher, it is configured to auto-terminate on re-launch which means that the platform should close itself and restart correctly. Together with Ctrl+B to build whole workspace (or Alt+B to build currently selected project) and re-launching the project with F11 can increase the development cycle time speed.

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. The `qL12000000` warnings can be safely ignored.

Between SoftConsole 6.0 and SoftConsole 6.2 the way how harts are started changed, now the servers do not use the auto-start feature, set PCs with the $SetAllPCs macro and only then invoke 'monitor start'. Previously the CPU errors on start could have been ignored, but now with a more correct approach there shouldn't be any errors. If Renode is printing CPU related errors, then this might point to a serious issue and shouldn't be ignored. One case where this might happen is when in the linker script's stack/heap size is not large enough, overflow occurs, and the CPU might get aborted because it returned to 0x0 location when it was exiting a function where the overflow happened. Make sure your application fits into allocated heap and stack sizes as the issues caused by overflows are hard to debug and can lead to catastrophic failures. See section Static stack profiling

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

When creating new launchers or porting older launchers from SoftConsole 6.0 then Environment settings needs to be added:
`RENODE_CUSTOM_SCRIPTS = ${env_var:SC_INSTALL_DIR}/renode-microchip-mods`


Name: PolarFire-SoC-Renode-emulation-platform

Environment variables to set:

Variable	Value
RENODE_CUSTOM_SCRIPTS	${env_var:SC_INSTALL_DIR}/renode-microchip-mods

[✓ Append environment to native environment]
Replace native environment with specified environment

Failing to do so might cause Renode to display the following error:

No such file C:\microsemi\SoftConsole\renode\bin/scripts/polarfire-soc-multiple-servers-base.resc.

There was an error executing command 'showAnalyzer mmuart0'

Received 'Peripheral not found: mmuart0' error while initializing analyzer for: mmuart0. Are you missing a required plugin?

After running the launcher verify if the platform and peripherals are loaded where they are expected by displaying all its peripherals, type into Renode's monitor window the following:

```
peripherals
```

On PolarFireSoC targets verify if the macros are loaded by typing:

```
runMacro
```

It should display which macros are bundled (for more information see the section Macros).

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/
- https://www.freertos.org/RTOS-RISC-V-SoftConsole-Renode-SiFive.html
- https://www.microsemi.com/product-directory/fpga-soc/5210-mi-v-embedded-ecosystem#renode-webinar-series

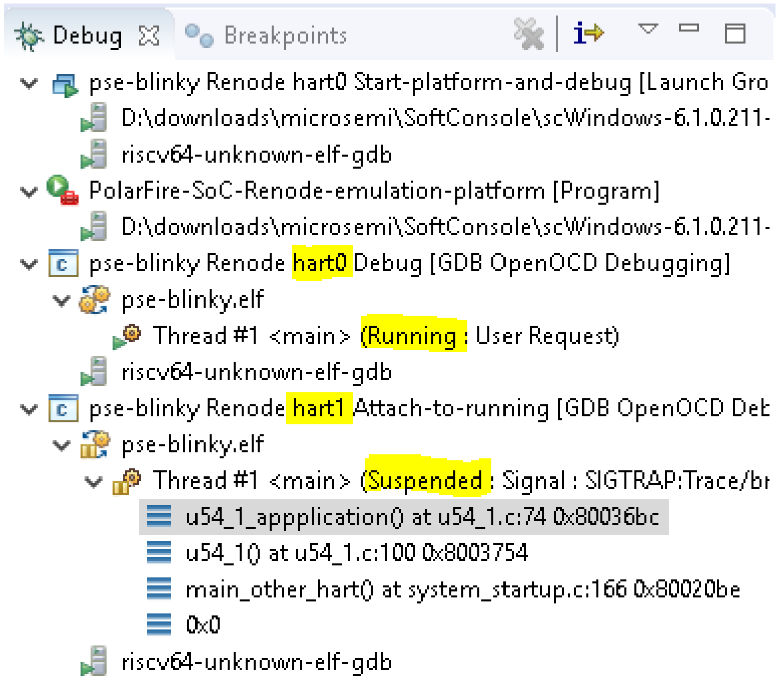
Multi-hart debugging

Example mpfs-blinky has two launchers which were preconfigured to demonstrate multi-hart debugging.

At first, the 'mpfs-blinky Renode all-harts Start-platform-and-debug' can be used to startup Renode, load the platform, load the application and start debugging. The 'mpfs-blinky Renode hart1 Attach-to-running' will connect to the running emulation platform, but expects the previous launcher to set the PCs already, loaded the executable ELF file and left the platform **running** (do **NOT** use it when any hart left the platform in halted/suspended state). This means that any other hart except the one which is debugged needs to be left running (Resume button, and preferably no breakpoints in that hart's code).

In contrast to a typical debug launcher, this type of launcher will not update the editor windows as the launcher "continued" the execution. The Suspend button must be clicked to halt/suspend the emulation after the connection was made (which does take while, watch the state of the `Window -> Show view -> Debug` to see when the second debugger is connected). This will halt the hart where it was currently executing code. It might be in some nested function (or loop) and the user might have to add a new breakpoint just to get into the desired location in the code. Setting u54_1 (or any other hart's main function) as the main breakpoint for these launchers might not work as all the harts started at the same time and the main breakpoint is very likely already passed even before the launcher tries to connect to the hart. Because the main was already passed and is not reachable will cause the session to try and break on unreachable breakpoints will make the GDB client indefinitely to wait for it.

When interacting with the harts extra attention needs to be paid to which hart is selected as the actions as send to that specific hart, see figure:



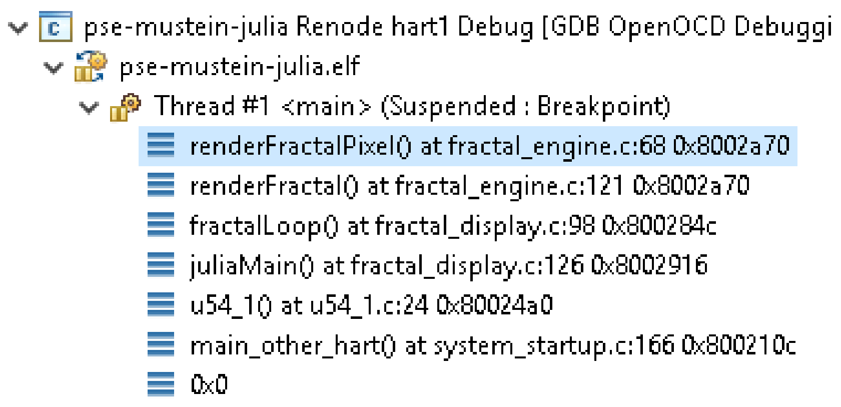
```
pse-blinky Renode hart0 Start-platform-and-debug [Launch Gro
    D:\downloads\microsemi\SoftConsole\scWindows-6.1.0.211-
    riscv64-unknown-elf-gdb
PolarFire-SoC-Renode-emulation-platform [Program]
    D:\downloads\microsemi\SoftConsole\scWindows-6.1.0.211-
pse-blinky Renode hart0 Debug [GDB OpenOCD Debugging]
    pse-blinky.elf
        Thread #1 <main> (Running : User Request)
    riscv64-unknown-elf-gdb
pse-blinky Renode hart1 Attach-to-running [GDB OpenOCD Deb
    pse-blinky.elf
        Thread #1 <main> (Suspended : Signal : SIGTRAP:Trace/br
            u54_1_appplication() at u54_1.c:74 0x80036bc
            u54_1() at u54_1.c:100 0x8003754
            main_other_hart() at system_startup.c:166 0x80020be
            0x0
    riscv64-unknown-elf-gdb
```

The u54_1_application() is selected, belonging to the hart1 connection, the hart1 is suspended and from the launcher it can be read that it's the hart1. If the desire is now to debug hart0 zero, first the Resume button must be clicked while the hart1 is selected, then the hart0 connection has to be selected and only then Suspend button can be pressed to debug hart0.

Steps required to make such launchers was described in Renode emulation platform section.

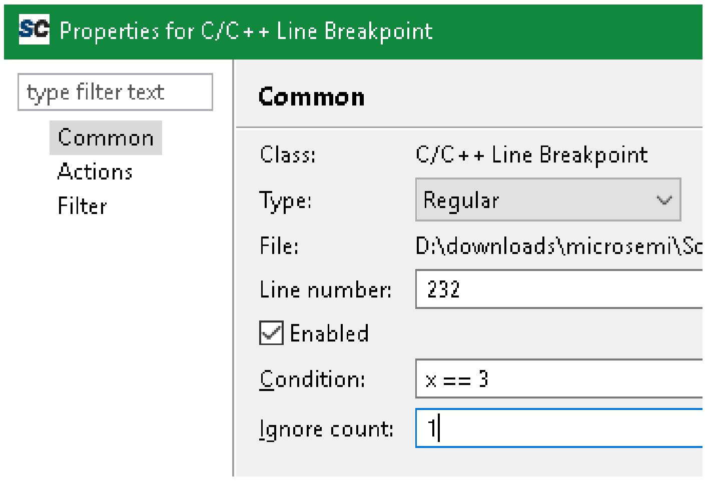
Call stack and conditional breakpoints

Normally call stacks can be ignored, but they are very useful when troubleshooting pre-existing conditions. When using the recommended workspace and its Develop and Debug perspective then this should be in the bottom-left section of the screen (it can be opened with `Window -> Show view -> Debug` in case it was previously closed)



```
pse-mustein-julia Renode hart1 Debug [GDB OpenOCD Debuggi
  pse-mustein-julia.elf
    Thread #1 <main> (Suspended : Breakpoint)
      renderFractalPixel() at fractal_engine.c:68 0x8002a70
      renderFractal() at fractal_engine.c:121 0x8002a70
      fractalLoop() at fractal_display.c:98 0x800284c
      juliaMain() at fractal_display.c:126 0x8002916
      u54_1() at u54_1.c:24 0x80024a0
      main_other_hart() at system_startup.c:166 0x800210c
      0x0
```

The figure above shows how nested the calls are, clicking on each will jump to the exact line of code from where the child stack was invoked and it will even show the content of local variables at that given time. This is very useful when troubleshooting what conditions lead to the given breakpoint. Using it with ISR handlers can show what the application was processing when the IRQ happened. Together with conditional breakpoints which makes debugging even easier. A conditional breakpoint can be enabled by holding CTRL and double-clicking on a regular breakpoint:



Class:	C/C++ Line Breakpoint
Type:	Regular
File:	D:\downloads\microsemi\Sc
Line number:	232
☑ Enabled	
Condition:	x == 3
Ignore count:	1

The figure above shows a conditional breakpoint which will break on a second event when variable x is equal to 3. The first event when this condition is met is ignored because ‘Ignore count’ is set to 1. This allows to create a complex condition where the code should break and together with call stack can be troubleshoot why and how this event happened. Debugging intermittent issues with these features is easier.

Note: This is not a Renode exclusive feature and can be used with HW targets as well.

Macros

Macros are useful in many scenarios:

- When using common and frequently used calls to remove code duplication (maintaining one macro is easier than maintaining all instances where it is used)
- When simplifying launchers, it is easier to invoke one macro than few commands
- When renaming actions, a command can be encapsulated into a macro in a case the original command is hard to remember.
- When a watch command is invoking few commands.
- When using 'monitor' with longer commands, at the moment the monitor command length is limited and might cause problems when invoking longer commands (they will get truncated and then they will misbehave). Macro can contain long commands and then it can be invoked with a short name.

SoftConsole v2021.1 has bundled a few macros, to get them listed run any PolarFire SoC platform (at the moment no Mi-V macros bundled) and type runMacro in the Renode's monitor window:

```
Available macros:
        global.BridgeNetworkMac0
        global.WiresharkRun
        global.MCsrLegend
        global.MCsrAll
        global.TraceAll
        global.LoadSymbols
        global.SetAllPCs
```

To invoke one of these the following syntax must be used:

```
runMacro $MCsrLegend
```

or

```
runMacro $MCsrAll
```

The MCsrAll macro can be used to troubleshoot trap related CSRs quickly on all harts without a need to use IDE. To see how they are made open the macro file:

```
<SC_INSTALL_DIR>/renode-microchip-mods/script/macros-pfsoc.resc
```

Any project including this file inherits the macros and it is possible for users to create their own.

When there is a need to trigger a command once per second, for example pressing a GPIO button, then a watch command can be used (Note: Copy/Paste might copy the wrong type of quotes, typing is safer):

```
watch "gpio0.button0 PressAndRelease" 1000
```

This will invoke the "gpio0.button0 PressAndRelease" command each 1000ms. If a CSRs needs to be fetched and monitored, then the watch command can just invoke a macro which can contain much more involved commands.

Combination of nested commands can be sometimes useful as well:

```
sysbus.u54_1 PC `sysbus.e51 PC`
```

The `sysbus.e51 PC` will be evaluated first, which will GET the value of the PC, it acts as GET because no second parameter with the value given after the PC property. Then this value is used as SET for the u54_1's PC, it acts as SET because after PC property there is a value given (the PC of the e51).

Note: Be aware when tracking MCAUSE and the IRQ is triggered (not a trap exception) that the mcause values can be greater than 0x8000000000000000 on 64-bit platforms and greater than 0x80000000 on 32-bit platforms.

Symbols and simple trace functionality

When debugging the debugging-client connects to debugging-server which then talks to the target.

With HW targets the GDB-client connects to OpenOCD (which implements and behaves as GDB server) and with FlasPro programmer talks through JTAG with the target.

However, Renode is overlapping slightly with the debugging-client task even when SoftConsole treats Renode only as the GDB server and target.

Typically, the symbol information is used only by the debugger-client, while the debugger-server nor the target do not need this information. Renode has some debugger-client features, but there is no automated mechanism to transfer these symbols from within SoftConsole as Renode is treated as regular GDB server (which typically doesn't need symbol information). This means that the following commands can't be automated in a generic manner.

The Renode must be launched, this is demonstrated on `mpfs-blinky` example, but can be applied to other projects.

Use the `mpfs-blinky Renode all-harts Start-platform-and-debug` group launcher, wait for it to break on main and load symbols into Renode by typing the following into Renode's monitor window:

```
sysbus LoadSymbolsFrom @../../extras/workspace.examples/mpfs-blinky/Debug/mpfs-blinky.elf
```

The path to the symbols is referenced relative to Renode's binary and needs to contain correct path to workspace, correct project, correct configuration (Debug/Release) and then correct binary name. Problem is that this is not generic and even when using macros, the path could be different depending on what workspace or what project name user's use. For PolarFireSoC users there is LoadSymbols macro, but requires users to set the SYMBOLS variable:

```
set SYMBOLS @../../extras/workspace.examples/mpfs-blinky/Debug/mpfs-blinky.elf
runMacro $LoadSymbols
```

These might be added to the debug launcher in some cases (with monitor prefix), but it's not a reliable way to load the symbols as the path might change and because monitor command has string length limitation (see section Macros).

Now the runMacro $McsrAll can resolve symbols and can tell the user where the PC of each hart is and what it's symbol. This can be used to resolve to what symbol some memory locations belong, for example (use backtick symbol):

```
sysbus FindSymbolAt `sysbus.u54_1 PC`
sysbus FindSymbolAt `sysbus.e51 MEPC`
```

The MEPC on target's boot-up is not populated and therefore not resolved, but when it is invoked later (when the emulation was started) it should get populated on the first interrupt or the first trap exception and then the resolution should work, this should be useful when troubleshooting traps as it can tell the user where the application had trap without a need to open the listing/assembly file and searching for the MEPC's value. See the section about trap exceptions: Progam has exited with code:0x00000003 (got into a trap exception)

It is possible to resolve all the PCs to symbols as the application runs. To enable it on e51 CPU, use the following command:

```
sysbus.e51 LogFunctionNames true
```

The LogLevel on the platform might be set to 'error' only and this is an 'info' level, therefore execute:

```
logLevel 1 sysbus.e51
```

On PolarFireSoC targets, there is 'TraceAll' macro which enables this trace feature on all CPUs, or 'TraceE51' macro if just E51 trace is needed (macros for all other harts exist as well). However, enabling too much of debug/trace information might cause unnecessary verbosity and make the debugging harder, therefore it's good to enable as little as needed.

Now running the mpfs-blinky application by pressing ‘Resume’ button (F8 shortcut) should start outputting simplified trace information. The SoftConsole’s Console tab should show when and what parts of the code was the CPU executing:

```
17:14:04.7767 [INFO] e51: Entering function MSS_UART_polled_tx_string at 0x80024D8
17:14:04.7767 [INFO] e51: Entering function gpio0_bit0_or_gpio2_bit13_plic_0_IRQHandler at 0x8003A1A
17:14:15.9247 [INFO] e51: Entering function gpio0_bit0_or_gpio2_bit13_plic_0_IRQHandler at 0x8003A1A
17:14:15.9378 [INFO] e51: Entering function gpio0_bit0_or_gpio2_bit13_plic_0_IRQHandler at 0x8003A1C
17:14:15.9488 [INFO] e51: Entering function gpio0_bit0_or_gpio2_bit13_plic_0_IRQHandler at 0x8003A1E
17:14:15.9603 [INFO] e51: Entering function gpio0_bit0_or_gpio2_bit13_plic_0_IRQHandler at 0x8003A22
17:14:15.9749 [INFO] e51: Entering function MSS_GPIO_set_output (entry) at 0x80032FE
17:14:15.9749 [INFO] e51: Entering function gpio_number_validate (entry) at 0x80034B8
17:14:15.9749 [INFO] e51: Entering function gpio_number_validate at 0x80034D4
17:14:15.9749 [INFO] e51: Entering function gpio_number_validate at 0x80034F6
17:14:15.9749 [INFO] e51: Entering function gpio_number_validate at 0x800350E
17:14:15.9749 [INFO] e51: Entering function gpio_number_validate at 0x8003524
17:14:15.9749 [INFO] e51: Entering function gpio_number_validate at 0x8003530
17:14:15.9749 [INFO] e51: Entering function gpio_number_validate at 0x8003552
17:14:15.9749 [INFO] e51: Entering function MSS_GPIO_set_output at 0x8003326
17:14:15.9749 [INFO] e51: Entering function MSS_GPIO_set_output at 0x800332A
17:14:15.9749 [INFO] e51: Entering function MSS_GPIO_set_output at 0x8003334
17:14:15.9749 [INFO] e51: Entering function MSS_GPIO_set_output at 0x8003362
17:14:15.9749 [INFO] e51: Entering function gpio0_bit0_or_gpio2_bit13_plic_0_IRQHandler at 0x8003A26
```

With the current release there is no way to use this data in a more visualized manner, but even in the text-form it can be extremely useful.

Sometimes it’s useful to build watch commands, for example to see what symbol the MEPC is when the IRQ/trap happened. It’s good to start building the commands in steps, first fetch the value of MEPC:

```
sysbus.e51 MEPC
```

Then resolving this address into a symbol (warping it with backtick character):

```
sysbus FindSymbolAt `sysbus.e51 MEPC`
```

And as last wrapping it into the watch command:

```
watch "sysbus FindSymbolAt `sysbus.e51 MEPC`" 200
```

This should keep updating the symbol where MEPC is pointing 5 times per second until Ctrl+C is pressed. If nothing is displayed then ‘Resume’ the application in the SoftConsole and trigger an IRQ by typing in Renode’s monitor:

```
gpio0.button0 PressAndRelease
```

Note: The user can make their own macros for Mi-V targets and use this feature however there are no macros bundled. Or use the commands directly without encapsulating them into macros.

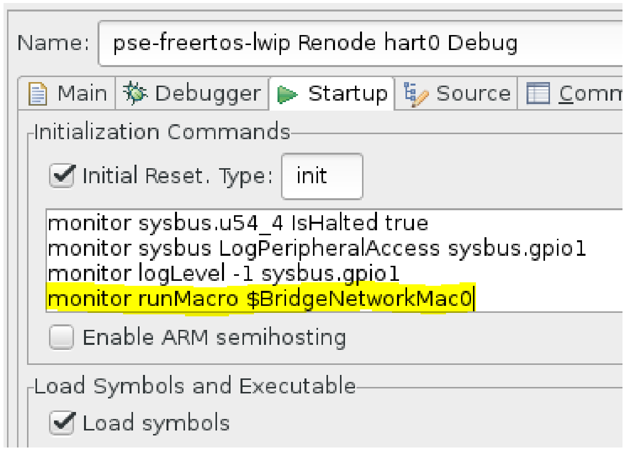
Creating bridge/tunnel to the emulation

The following section will cover necessary steps to access the emulated network from within host OS, however in SoftConsole v2021.1 this feature is accessible only from Linux hosts.

Currently the mpfs-freertos-lwip example is only SoftConsole's example where this feature can be demonstrated. To access the emulated network the following macro must be added (there might be already preexisting monitor commands) to the debug launcher's (in this case `mpfs-freertos-lwip Renode all-harts Debug`) section `Startup -> Initialization`:

```
monitor runMacro $BridgeNetworkMac0
```

The bridge is sensitive if it was invoked before or after the emulation started and therefore keeping it in section `Startup -> Initialization` will be the most reliable way to use it:



Name: pse-freertos-lwip Renode hart0 Debug

Initialization Commands

Initial Reset. Type: init

```
monitor sysbus.u54_4 IsHalted true
monitor sysbus LogPeripheralAccess sysbus.gpio1
monitor logLevel -1 sysbus.gpio1
monitor runMacro $BridgeNetworkMac0
```

Enable ARM semihosting

Load Symbols and Executable

Load symbols

When creating the bridge for the first time it might display error:

```
Could not set TUNSETIFF, error: 2
```

Renode is trying to probe user's privileges, which should be safe to ignore. If the polkit and sudo are set to be password-less, then the bridge should be created correctly without any other user's interaction.

See the chapter: "Could not set TUNSETIFF, error: 2"

Then as superuser (root user, or regular user with sudo privileges) invoke on a Linux terminal:

```
ifconfig renode-tap0 172.16.0.1/24 up
```

For best experience copy/paste this command from the example's UART output as it is updated dynamically depending on how the network is set up, while the command above only covers the default settings.

Make sure that the renode-tap0 network is not overlapping with any other network on the user's host. Including the virtual networks and bridges like docker's network. When targeting real HW it is desired to make it part of an existing network (overlap with existing networks) and just keeping the IP unique (do not create conflicts on the network). However, for the way how Renode is making the tunnel it is best to have completely unique and separate networks while the host's IP will be the targets gateway. If this is not satisfactory and overlapping is required then to make it work might require other extra steps which are not covered in this document (setting up routes, gateway, firewall, software bridge …)

By default, the target's IP address and mask are `172.16.0.3 / 255.255.255.0`

These settings can be change from within this file:

```
/src/modules/config/lwip-2.0.0-wip/network_interface_settings.h
```

Now running `ping 172.16.0.3` should start responding, however this might not be true for other applications, as they might have ICMP or even ARP disabled (see all troubleshooting sections related to networking). If ping support is undesired then it can be disabled easily from:

```
/src/modules/config/lwip-2.0.0-wip/lwipopts.h
```

By changing `LWIP_ICMP` define value the ping can be disabled, other networking features and debugging verbosity can be tweaked from this file as well.

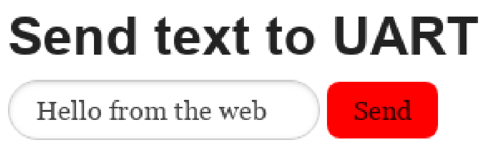
The emulation is not as fast as the real hardware therefore the ping response times might vary:

```
# ping 172.16.0.3
PING 172.16.0.3 (172.16.0.3) 56(84) bytes of data.
64 bytes from 172.16.0.3: icmp_seq=1 ttl=255 time=202 ms
64 bytes from 172.16.0.3: icmp_seq=2 ttl=255 time=3.13 ms
64 bytes from 172.16.0.3: icmp_seq=3 ttl=255 time=2.43 ms
64 bytes from 172.16.0.3: icmp_seq=4 ttl=255 time=4.74 ms
^C
--- 172.16.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 2.432/53.206/202.517/86.208 ms
```

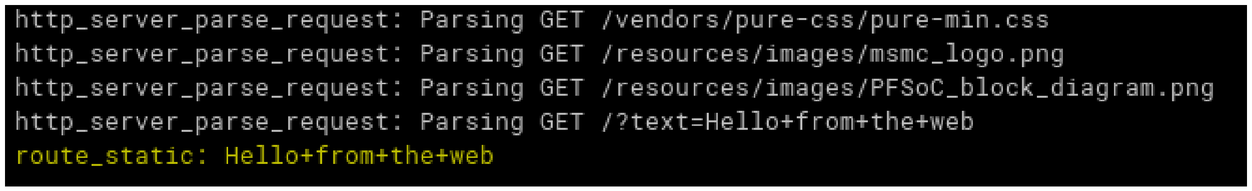
The first ping might be extra long as other tasks had to be finished first (such as ARP address exchange)

Then with a generic webserver browser opening the location: `http://172.16.0.3`

Should open the demo website running from within target. Scrolling down to the page to the section “Send text to UART”, entering text into the HTML form and pressing “Send” button:



Then the firmware should get the text displayed within the UART’s output:



```
http_server_parse_request: Parsing GET /vendors/pure-css/pure-min.css
http_server_parse_request: Parsing GET /resources/images/msmc_logo.png
http_server_parse_request: Parsing GET /resources/images/PFSoC_block_diagram.png
http_server_parse_request: Parsing GET /?text=Hello+from+the+web
route_static: Hello+from+the+web
```

Feel free to experiment with the current example but note that currently bundled HAL is not most up-to-date with the recent HAL release, the GEM driver required few changes to be made on top of HAL so it is not straight forward to port to the newest HAL. This will be addressed in a forthcoming release, however in the meantime use the bundled mpfs-blinky, mpfs-mustein-julia or firmware catalog examples which use the most recent HAL.

Related sections:

Using Wireshark to monitor the network traffic inside the

Web content of the mpfs-freertos-lwip example

How do I delete Renode’s bridge TAP networking interface?

Can’t see the interface after invoking BridgeNetworkMac0 macro

Wireshark shows a significant amount of traffic and overloads my target

Using Wireshark to monitor the network traffic inside the emulation

In SoftConsole v2021.1 release this feature is accessible only from Linux hosts. The following steps are mainly covered from Ubuntu perspective, for other distributions contact your system administrator or execute the equivalent steps for your distribution yourself.

To startup Wireshark with each debug session open the mpfs-freertos-lwip example, edit the `mpfs-freertos-lwip Renode all-harts Debug` launcher and in the `Startup -> Initialization Commands` text-area box append the following command: `monitor runMacro $WiresharkRun`

Launching the group launcher now should open Wireshark windows. If this is not happening, try to invoke `runMacro $WiresharkRun` command from within Renode's monitor window. If Renode prints error "Wireshark is not installed or is not available in the default path", if the Wireshark is installed by the package manager (and not built from source) then it should be in the path already. Double-check if Wireshark is installed on the system and where it is by typing on the Linux console terminal:

```
whereis wireshark
```

If Wireshark is found but it's not in the path, then add it to the path temporary with:

```
export PATH=<WIRESHARK_PATH>:$PATH
```

And then start SoftConsole from the same terminal.

If it's not installed, then type the following to install it:

```
sudo apt-get install wireshark
```

Answer YES, if user will be asked if "`Should non-superusers be able to capture packets?`".

If this dialog was not shown it's possible that the package already was installed once and configured and then uninstalled. Proceed to completely remove the package with its settings. The 'remove' option only removes the application and leaves configurations files untouched, while the 'purge' option removes configuration as well. Invoke the following command:

```
sudo apt-get purge wireshark
```

After a complete removal, proceed to install it back again, now the dialog question should be displayed.

If Wireshark now when launching the launcher opens but displays popup error "`Couldn't run /usr/bin/dumpcap in child process: Permission denied",` then verify the file privileges:

```
ls -la /usr/bin/dumpcap
```

If it is accessible to root user and Wireshark group, then most likely the current user is not added inside the Wireshark group. The following command should add user to Wireshark group:

```
sudo usermod -a -G wireshark <YOUR_USER>
```

For the group changes to take effect typically logging out and logging back to the user's account should be enough, however sometimes reboot is necessary.

If everything is configured properly now the Wireshark should work and run automatically on each launch. The Wireshark is sensitive if it was invoked before or after the emulation started and therefore keeping it in section `Startup -> Initialization` will be most reliable.

As mention in other sections, the emulation is not running at full speed of real hardware and under heavy load, some packets might be dropped/timeout. Depending on the application/protocols (and firmware's IP stack) the packets might get retransmitted transparently to the end-user.

Related section: Wireshark constantly shows dialog about unsaved packets

Emulation time

The host's real time and the emulation time might not be passing at the same speed, the ratio itself is very dependent on the host's performance, platform settings and the application itself.

How many instructions per second can be emulated depends on the host's performance, OS, target application and the settings. The emulation time speed is mostly depending on the performance of the host and the guest's MIPS rating. There are other factors such as the amount of the cores the emulated platform has. A very significant factor is how the application is written and how much time cores spend in sleep or are halted (See chapter Tweaking the emulation performance). In a case where emulator can spend most of the time in sleep will not require as many instructions to emulate for a given time to pass, it's easier to wait for the next interrupt instead of looping inside an infinite loop. The RISC-V CPUs (32-bit and 64-bit) have default 64MIPS rating, if required this can be changed from Renode's platform script or SoftConsole debug launcher (in launcher scrip the commands need the `monitor` prefix).

To set 20MIPS rating on PFSoC's CPUs add the following to your platform script:

```
sysbus.e51 PerformanceInMips 20
sysbus.u54_1 PerformanceInMips 20
sysbus.u54_2 PerformanceInMips 20
sysbus.u54_3 PerformanceInMips 20
sysbus.u54_4 PerformanceInMips 20
```

Note: If editing the <SOFTCONSOLE_DIR>`/renode-microchip-mods/scripts/polarfire-soc-multiple-servers.resc` file then be aware these changes will be applied on all projects which are using this platform and in general is not recommended. The recommended way is to change platform script files when the project has its own dedicated platform script file. It is possible to copy existing scripts, or even include existing platforms and only do changes on top of the included generic platform (see files in the /renode-microchip-mods/scripts/). When adjusting settings for a single project while using the existing bundled platform, then changing the debug launcher is the preferred method (commands need `monitor` prefix).

If a higher MIPS rating is given to the platform than the host can handle in real-time will cause the emulation time to pass at a slower rate than the time is passing on the host.

To check what MIPS settings are set to e51 CPU (e51 is used just as an example):

```
sysbus.e51 PerformanceInMips
```

To check how much instructions e51 are executed:

```
sysbus.e51 ExecutedInstructions
```

Note: This value might be misleading, preferable is to have as many MIPS executed as possible but wasting these MIPS on infinite loops is not gaining any performance and on contrary might cause the virtual time to pass at a slower rate. Spending as much time as possible in the sleep will enable the emulator to be more efficient and emulate more of the emulation time than it would be able to when all harts would be too busy cycling in infinite loops. Therefore, sometimes a lower ExecutedInstructions can achieve better performance. Having a host capable to handle as many MIPS as possible is good, but as important is to not waste these MIPS emulating unnecessary and wasteful instructions.

To see how much of emulation time passed:

```
machine ElapsedVirtualTime
```

To check every second how much of the emulation passed:

```
watch "machine ElapsedVirtualTime" 1000
```

Note: If any of these needs to be part of a debug launcher then be aware of the monitor command string length limitation. If there is a need to invoke longer commands from the launcher then using variables and making custom macros could be a workaround. See the "Time framework" chapter of Renode's documentation and the Release Notes section: Macros

Tweaking the emulation performance

If emulating multi-node (multi-board) platform then the time ratio between the nodes will not be affected. When there are server and client set up in the platform and a network request with a 1second timeout is sent. Then the timeout in the emulation will be timed as expected no matter if that took on the host 5seconds or 0.5 seconds to emulate it.

If there is the need to run real-time communication with the host's network then some tweaks to the platform and application need to be made. When overloading the slower emulated target with many requests then the target might not be able to respond within the required time, however few steps could improve this situation:

- Do not send unnecessary traffic, monitor with Wireshark and remove sources of any traffic which is undesired

- If latencies are not critical then do not pool peripherals constantly in a loop, give it a delay or put the core into the sleep (if there will be timer interrupt happening soon).

- If using custom models, do refactor them and apply good coding guidelines to make them more efficient. For example, do not use many "if" conditions on register reads/writes, instead use the `DoubleWordRegisterCollection` lookup table.

- Spend as much time as possible in a sleep state, put CPU into sleep with WFI instruction, use inline assembly:

  ```
  __asm("wfi");
  ```

- Completely halt unneeded harts (revise the HAL's constrains of this approach mentioned in sections above). They can be halted from within the platform as done in the mpfs-mustein-julia demo (as the demo has its own customized platform bundled with the project):

  ```
  sysbus.u54_4 IsHalted true
  ```

  Or they can be halted from the debug launcher as done in the mpfs-blinky or pse-freertos-lwip:

  ```
  monitor sysbus.u54_4 IsHalted true
  ```

- Optimize the firmware itself. When targeting 64-bit systems there are common mistakes which developers are more likely to introduce to their code. One of which is not using the native size of the platform. Using 32-bit loop counter still can occupy whole 64-bit register but it will do extra instructions to make it behave as a 32-bit register. While using 64-bit integer on a 64-bit will translate directly and natively to a single register without any unnecessary instructions. Use static code analysis tools and refactor problematic code sections.

For example, by default the FreeRTOS idle task is just an infinite loop, but there is a way to implement own idle task:
https://www.freertos.org/RTOS-idle-task.html
Set `configUSE_IDLE_HOOK` to 1 in `FreeRTOSConfig.h` and implement sleep in the idle hook:

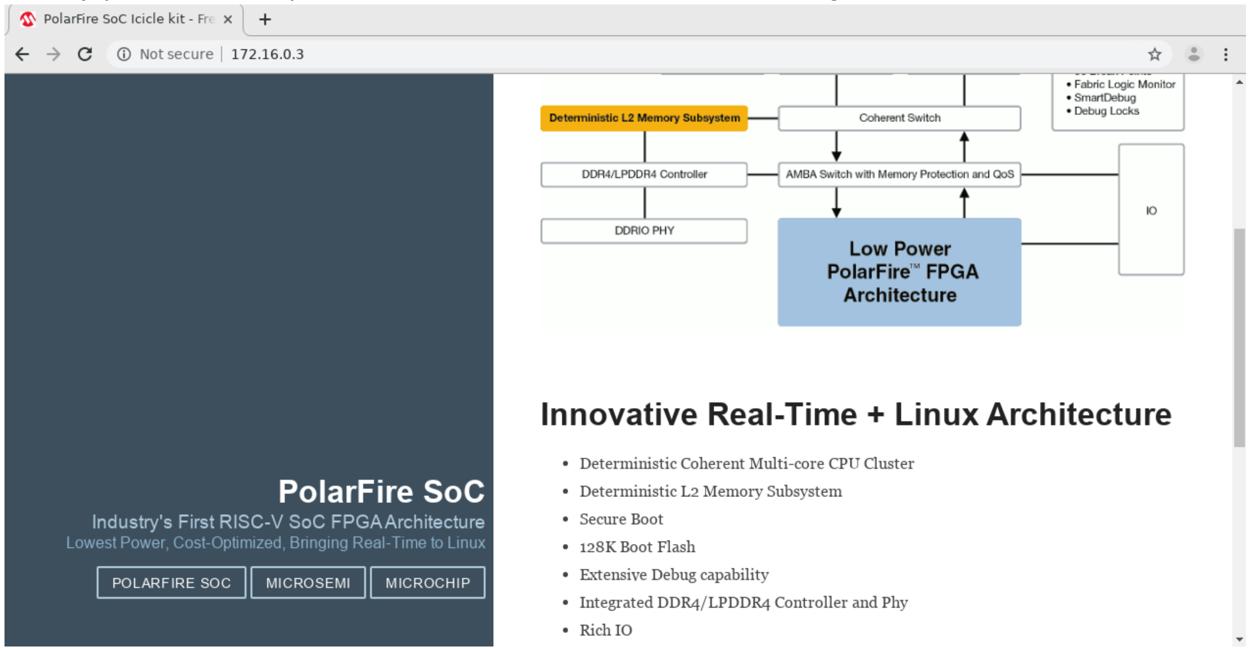
```
void vApplicationIdleHook( void )
{
   __asm("wfi");
}
```

This change can show even 20-fold difference between the FreeRTOS with sleep and without sleep. Meaning that fewer instructions were needed to be emulated for the same time on the guest to pass.

Web content of the mpfs-freertos-lwip example

This bundled example is showcasing how to implement a FreeRTOS with a LwIP networking stack running a webserver on the PolarFireSoC Icicle kit emulation platform. This is just an example, has only minimal implementation of a webserver and shouldn't be used for production code.

Previous sections covered how to bridge the network and interact with the target, when these steps are executed correctly (on the Linux host), then the webserver should be accessible from a generic web-browser:


PolarFire SoC Icicle kit - Fre
Not secure | 172.16.0.3
Fabric Logic Monitor
SmartDebug
Debug Locks
Deterministic L2 Memory Subsystem
Coherent Switch
DDR4/LPDDR4 Controller
AMBA Switch with Memory Protection and QoS
IO
DDRIO PHY
Low Power PolarFire™ FPGA Architecture
PolarFire SoC
Industry's First RISC-V SoC FPGA Architecture
Lowest Power, Cost-Optimized, Bringing Real-Time to Linux
POLARFIRE SOC
MICROSEMI
MICROCHIP

Innovative Real-Time + Linux Architecture

- Deterministic Coherent Multi-core CPU Cluster
- Deterministic L2 Memory Subsystem
- Secure Boot
- 128K Boot Flash
- Extensive Debug capability
- Integrated DDR4/LPDDR4 Controller and Phy
- Rich IO

To change the content of the webserver the routes need to be bonded to the new page callback handlers, see:

`/src/application/web-server/routes.h`

The callbacks (route handlers) themselves are typically stored in:

`/src/application/web-server/route_handlers.c`

To respond differently depending on the URL's query, a `http_request_get_query_value(req, "query_key")` call can be used. In this example it is used to send messages to UART (see `route_static` function). Callbacks can produce the response directly, use simple string replacement template engine (see: `route_template_index` function) or read the content from the embedded payload (see: `route_static` function). To change the embedded payload data these three steps must be followed:

1. Edit the content in the `/src/application/embedded_autogenerated_data/data_root_raw` folder only (all other files in the `/src/application/embedded_autogenerated_data` are autogenerated)
2. Run the `mpfs-freertos-lwip Embed-the-web-content` external launcher to generate the hex-dumps and metadata required for the payload to be bundled. The tool can be used from Windows/Linux command line, can be part of Continuous Integration or be part of a Makefile projects.
3. Rebuild the project (partial/incremental build should be sufficient)

The external launcher will compress html/htm/js/css/xml/json/wasm files and they can't be used directly without decompression, this example webserver is only capable of passing through the compressed data. However the GZIP compression is used and virtually all browser from the past decade support transparent decompression feature (https://webmasters.stackexchange.com/questions/22217 and https://en.wikipedia.org/wiki/HTTP_compression). On embedded targets with restrained storage capacities, this is a useful feature as it can reduce the static content size to 10-20% of its original size. Just remove the `-compress_web=true` argument from the external launcher in cases where this feature needs to be disabled. The `route_template_index` function uses `.tpl` file extension postfix to avoid compression on that single specific file only. For more information and argument options see:

`<SOFTCONSOLE_DIR>/extras/ead/README.pdf`

Tips, tricks, and issues

Using Open Source Software tool such as Renode will contaminate my proprietary code?

Using the tool will not impose any obligations or restrictions on the user’s code, his license or his IP.

In an edge case where the user would copy part of the Renode’s source code, modify it and redistribute its binaries, then there are obligations to meet, however Renode has MIT license and allows wide use (including commercial, private use and sublicensing it with a more restrictive license):

https://tldrlegal.com/license/mit-license

If in doubt consult with a legal team.

There was an error executing command 'sysbus.cpu StartGdbServer 3333 true'

Something is keeping the port 3333 opened, it could be some other application, but in most cases it’s just a previous instance of the Renode. Close it by pressing X on the window or stop the group launcher which should stop the Renode and the gdb client as well.

Wireshark constantly shows dialog about unsaved packets

1. Wait for the Renode’s Wireshark to open
2. Go to Edit -> Preferences
3. Uncheck the “Confirm unsaved capture files”
4. Click OK (user’s Wireshark settings are not affected as these settings are for Renode’s Wireshark)

Wireshark shows a significant amount of traffic and overloads my target

It is important to distinguish between really lost packets (for example when buffer overflow happened) and packets which just took too long for a response that they were considered lost and re-transmitting was triggered.

In the case of PolarFire SoC model a Warning level logger can be enabled on the CadenceGEM peripheral by using:

```
logLevel 2 sysbus.mac0
```

If a buffer overflow occurs, then Renode will print the following message:

```
Receive DMA buffer overflow
```

Because Renode will not emulate the PolarFire SoC model at full speed, therefore the performance cannot be compared with the real hardware. Processing packets on Renode and sending responses will take longer than they would on the real hardware. Networking is very latency-sensitive and many protocols can timeout and send a retransmission of the request while the target was processing the response, causing the target to have even more requests to process. Therefore extra care should be made to not overload the target with unnecessary packets.

It’s recommended either not to bridge the renode-tap0 with the host’s interface to keep away unnecessary traffic and keep the interfaces separate with their own, non-overlapping IP networks. Even if the interfaces are not bridged there might be daemons running which queries and discovery requests to all networks. For example, NTP time sync service might at some cases start querying the network range and produce frequent packets. Temporarily disable the daemon to see if it does improve the issue:

```
sudo /etc/init.d/ntp stop
```

Monitoring what process do on the network might give indications of what services/tools might be causing unnecessary packets. Use “`netstat`” to print networking connections and disable unneeded services which flood the Renode’s network with unnecessary packets and requests.

“Could not set TUNSETIFF, error: 2”

When creating a networking bridge from the Renode emulation a TUN/TAP interface on the host must be created. Renode probes if the interface is already present and when it’s not, it will try if a current user has privileges to create such interface. In most cases, the current user doesn’t have privileges to do so and will invoke this error message. In general, it should be safe to ignore this error as it is sometimes expected.

When the user has no privileges to create the tap, then Renode will invoke polkit to elevate users privileges for this command to succeed. Behavior differs between distributions and how polkit/sudo is configured. It might request for user’s password, it might continue without prompt or it might wait for a fingerprint scan.

If the tap interface already is present on the system, Renode should continue without error message even when previously it was displaying the message.

Related topics in Renode’s documentation:

https://renode.readthedocs.io/en/latest/networking/host-network.html

https://renode.readthedocs.io/en/latest/networking/wireshark.html

How do I delete Renode’s bridge TAP networking interface?

As a superuser:

```
ip link del tap0
```

As a regular user:

```
sudo ip link del tap0
```

Can’t see the interface after invoking BridgeNetworkMac0 macro

The behavior strongly depends on the distribution’s settings (configuration of `/etc/sysconfig/network-scripts/` in CentOS/RHEL or `/etc/network/interfaces` in Ubuntu/Debian). However, in many cases the interface is by default down and will only show in the list when “`ifconfig -a`” is invoked. The interfaces needs to be brought up and probably with a specific IP, for example:

```
sudo ifconfig renode-tap0 172.16.0.1/24 up
```

The host IP shouldn’t be in conflict in any IPs on the network. Preferably your host IP should be guest’s gateway IP. The mask settings should match with mask settings of your target. The /24 is an equivalent representation of the 255.255.255.0 mask.

Can’t ping my target

There might be a networking issue (go over the other existing networking related sections), or it might be because the target doesn’t have ICMP (ping) service/support. Not all targets do implement ping and the networking might be working correctly even without working ping.

The `mpfs-freertos-lwip` example does support ping and ICMP protocol and it’s by default enabled with the `LWIP_ICMP` define in the `/src/modules/config/lwip-2.0.0-wip/lwipopts.h` file

Check the section below to troubleshoot if the target is responding to any packets at all:

My target doesn’t respond to packets

It can be caused by many various reasons and might be different between the host’s settings and as well different between the applications themselves as they can have different IP stacks (which can be configured differently).

- A firewall is blocking the packets, temporarily disable the firewall. In case your distribution does save iptables on the reboot, then it’s preferable to save the rules before deleting and then restoring the rules before rebooting:

  ```
  sudo iptables-save > <TEMPORARY_FILE>
  ```

  And to restore them back:

  ```
  sudo iptables-save < <TEMPORARY_FILE>
  ```

  If a `netfilter-persistent` package is installed, then the following should work:

  ```
  sudo service netfilter-persistent save
  ```

  Package such as `iptables-persistent` might be restoring the settings every single time. To delete all existing firewall rules and have only “accept” policy issue as super user the following:

  ```
      iptables -P INPUT ACCEPT
      iptables -P FORWARD ACCEPT
      iptables -P OUTPUT ACCEPT
      iptables -t nat -F
      iptables -t mangle -F
      iptables -F
      iptables -X
      iptables -S
  ```

- The target is not capable to respond to ARP messages. When using cut-down IP stack without ARP feature, then on the host the target’s HW address has to manually inserted into the ARP table:

  ```
       sudo arp -s <TARGET_IP> <TARGET_MAC>

       sudo arp -s 172.16.0.2 00:fc:00:12:34:56
  ```

- Browsers might overwhelm the target as they can query many requests concurrently (such as favicon), instead of a browser try wget:

  ```
      wget http://172.16.0.2
  ```

  Or use `curl` which is more flexible and allows to control headers of POST/PUT requests as well.

- If all other interfaces were put down, then the renode-tap0 can be made to the default gateway:

  ```
      sudo route del default                # delete previous default gateway
      sudo route add default gw 172.16.0.1  # add your default gateway
      sudo route -n                         # show routing table
  ```

- The Renode’s networking tunnel is similar to a tap, for some users this might be useful:
  http://www.shakthimaan.com/installs/debian-tun-tap-setup.html

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. Depending on the nature of the application these might not be as predictable and might require an engineer to evaluate the application manually and manually decide on the sizes. 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/

https://dzone.com/articles/gnu-static-stack-usage-analysis

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

https://stackoverflow.com/questions/6387614

https://www.adacore.com/uploads/technical-papers/Stack_Analysis.pdf

The -fstack-usage can show what parts consume how much stuck, when run on mpfs-mustein-julia example it can show in the file fractal_display.su (the static usage has .su extension) that one function is allocating a non-negledable amount on the stack (64KiB):

```
fractal_display.c:70:6:transition      160    static
fractal_display.c:84:6:fractalLoop     80     static
fractal_display.c:115:6:juliaMain      65568  static
```

Opening the fractal_display.c will show that there is a buffer allocated (pixel buffer 128 pixels by 128 pixels big, with 4 bytes per pixel):

```
void juliaMain(uint64_t base)
{
          uint32_t buffer[SCREEN_WIDTH * SCREEN_HEIGHT];
```

This either has to be refactored and removed from the stack or it needs to be considered inside the linker script stack allocation. Moving some buffers from local (stack) to the global scope (bss/data) will make their size predictable as there will be guaranteed only one instance present. While keeping them locally might cause multiple allocations if the function is invoked recursively. Sometimes this is necessary and each nested invocation needs their own instance of the variable and sometimes moving variables to globals can be considered as a code smell, therefore blindly moving local variables to global scope might create more problems and bugs. Consider all the pros and cons of each section of code and make decisions case by case.

The size of variables has an impact, but their nesting as well, recursive algorithms might take a significant amount of iterations and allocations on the stack (even if they are using small variables). Many application and algorithms by their nature cannot by predicted statically with 100% confidence and depend significantly on their run-time behavior. Therefore the analysis tools should be considered as guiding helpers which can show a problematic place in code but can't be blindly trusted.

Other tools such as LDRA, cppcheck (but not limited to) could be used to analyses the code and spot some potential issues. Sometimes when stack size can't be increased it might be possible to refactor the code and conserve the stack allocations as much as possible.

If the C or C++ application is allocating variables dynamically, then heap size should be evaluated as well:

https://www.learncpp.com/cpp-tutorial/79-the-stack-and-the-heap/

Related sections:

Renode emulation platform

Deeply nested code causes crashes and glitches

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`.

Cppcheck was not intended to replace all analysis tools so it is recommended to use other analysis tools (not bundled with SoftConsole) as well.

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

The underlying RISC-V GNU toolchain does support C++ and SoftConsole is supplied with one C++ example, but RISC-V C++ projects have not been extensively tested within the SoftConsole Eclipse/CDT environment. Using C++ with embedded targets needs to considered carefully and when making C++ project then only a subset of the C++ language should be enabled (see the bundled example what options/features of the C++ the example disabled). Useful references:

http://www.open-std.org/jtc1/sc22/wg21/docs/TR18015.pdf

https://codereview.stackexchange.com/questions/151275/simple-c-alternative-to-exceptions-for-embedded-systems

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.

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.

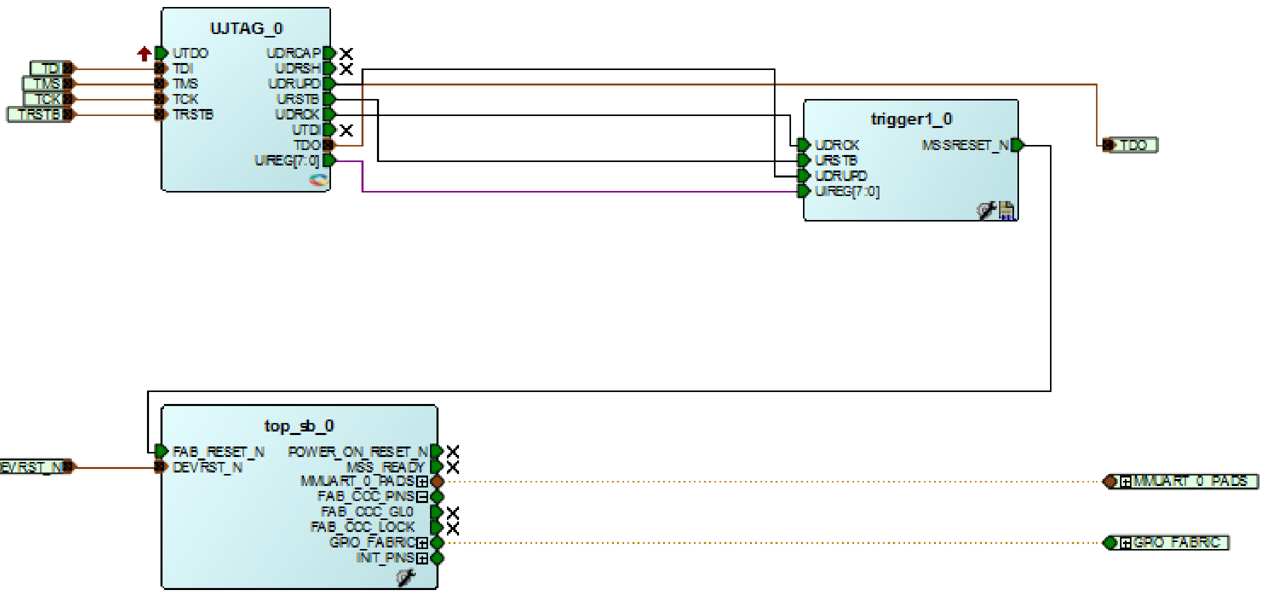
SmartFusion2 DPK unlocking

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
```

DPK unlocking only works if the MSS is reset after the DPK unlock operation has been executed and this does not happen automatically. For DPK unlocking to work reliably some additional logic should be added 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
# $DEVNAME environment variable is most likely set to /dev/ttyUSB0
# get the ttyUSB0 without the path
NAME=$(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. Note the newlib doesn't have fully ported floating point support for RISC-V and is not utilizing some floating point instructions even if they are supported by the hardware (fmin/fmax/fsqrt…) and some functions might accidentally include soft double support. This can lead to the application footprint growing unexpectedly. If really needed, it's possible to avoid these newlib functions and replace them with own implementations using inline assembly.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. See the miv-rv32imaf-raytracer-uart-cpp as an example when making a C++ project. C++ is a large language with an overhead which is very significant from an embedded perspective. Disable as much as is viable in the Optimization tab settings:

   "Do not use exceptions", "Do not use RTTI", "Do not use _cxa_atexit()" and "Do not use thread-safe statics".

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 (got into a trap exception)

Applies to any other exit codes. And this section can be used on other trap exceptions as well (they do not have to display a error on the UART)

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
```

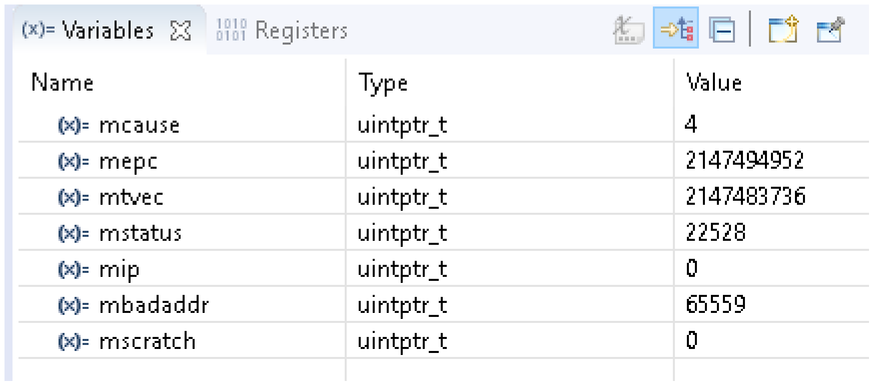
The exit code is the RISC-V Privilege Specification `mcause` register value plus literal 1:

```
exit_code = mcause    + 1
mcause    = exit_code - 1
```

The exit code table above can be used to troubleshoot `mcause` values as well, just subtract 1 from the exit_code value.

For full information refer to the draft RISC-V Privileged ISA Specification: https://riscv.org/specifications/privileged-isa/

Get the value of `mepc` CSR register, it points to memory where the trap happened. Create breakpoint on this location. Single-step (if needed) to the moment right before the trap happens, note the current state of the CPU, then let the trap happen and 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 your problem, **do not ask for help without supplying this vital information!** Our Mi-V verbose trap handler when running Debug build (Release builds do have it disabled) should show the values as local variables in the bottom right corner (however it's not the only method how to fetch them):



Name	Type	Value
(x)= mcause	uintptr_t	4
(x)= mepc	uintptr_t	2147494952
(x)= mtvec	uintptr_t	2147483736
(x)= mstatus	uintptr_t	22528
(x)= mip	uintptr_t	0
(x)= mbadaddr	uintptr_t	65559
(x)= mscratch	uintptr_t	0

Note: Converting these values to HEX should give more meaningful representation (Right-click -> Format Number -> Hex)

These CSRs have the following meanings which should help troubleshoot the cause of why the exception happened:

mcause Should be the same as exit code -1, if it's not then different trap happened in middle of the trap debug session.

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

mtval/mbadaddr Depending on the assembler version the name of this CSR might be either mtval or mbadaddr. In the current SoftConsole v2021.1 these can be freely interchanged, however the mtval is the newer correct name. If the exit code is 1,2, 5, 6, 7, 8, 0xE or 0x10 (mcause 0,1,4,5,6,7,0xD,0xF) then this CSR points to the memory address which caused this fault. If the exit code is 3 (mcause 2) then this CSR contains the opcode of the instruction which triggered the trap. Together with the exit code (mcause) and mepc 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.

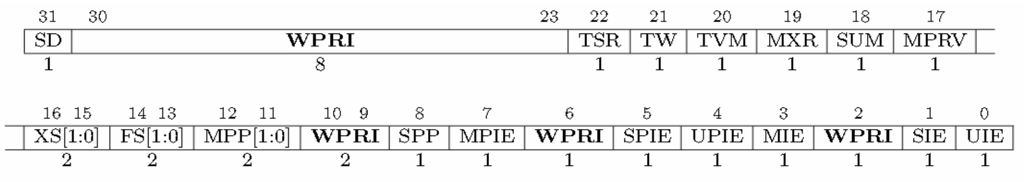
mip Displays any pending interrupts.

mie Displays what interrupts are enabled.

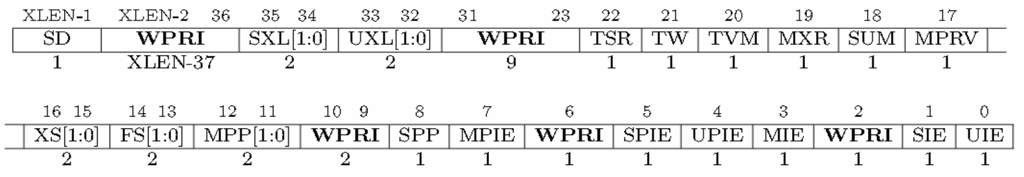
mtvec Is trap handler vector, address where to jump when the trap happens.

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 (RV64/RV128):


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

Note: Depending on the application and the problem, but typically the most useful and important CSRs are mcause, mtval/mbadaddr and mepc. Therefore, when troubleshooting a trap issue these 3 CSRs should be addressed first as they can give very good picture what happened and why.

Example: When mcause=2 then look at mtval/mbadaddr to see if the opcode of the instruction is not full of zeroes or ones, which would indicate that the code went executing the wrong region of memory (troubleshoot why it might have jumped there). Look at the mepc value which will point to the location address where the problem occurred. Open the listing file (.lst) and search for that address, this should point a specific instruction, validate if the instruction is valid. If it has compressed syntax (instructions have c. prefix and the opcodes are only 16-bit), then check if your HW has support for the C extension and if the project settings have correctly configured the CPU/arch/abi. A similar situation is with F and D floating-point extensions. The listing file will show symbol to which C function the instructions belongs and opening the function from the C editor might reveal a potential issue or narrow down the scope of potential causes. **It is easier to ask for assistance when the possible causes are narrowed down.**

Example: When mcause=0, 4, or 6. Then this might indicate misaligned address access, the mtval/mbadaddr show the address itself (if it's not aligned to 32-bit alignment then it's misaligned) and mepc should point to the location in the code where this misalignment happened, evaluate the listing file to confirm the issue. Double-check project settings: Use strict alignment

If the target is Renode, then it is possible to use symbols to resolve the addresses such as MEPC which can sometimes help troubleshoot a trap without even opening the listing file, see Symbols and simple trace functionality

Size optimizations cause trap exception: Load/Store address misaligned (mcause 4/6)

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)

See section: Use strict alignment

<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 6.1 (and newer) the GTK2 support is removed and it might cause incorrectly rendered UI elements in some cases and on some distributions even unstable UI. Distributions such as RHEL 6.x and CentOS 6.x which provide only GTK2 are not supported anymore. These distributions do not provide GTK3, unless the user is allowed and able to compile and install all dependencies and whole GTK3 subsystem on their RHEL 6.x / CentOS 6.x machines, then they will not be able to use SoftConsole 6.1 without issues. Second workaround is not trivial as well which involves installing Docker 1.7.1 (the highest version which can run on such legacy kernel, unless user wants to recompile the Docker from sources). And installing SoftConsole inside docker container (Debian9 was tested), it is possible to tunnel the X11 application and share the host's FlashPro5. Running a VM with newer OS can be used only if Renode is targeted, but it will not allow user to interact with real HW as the FlashPro5 driver will not work inside the VM. An upgrade of the OS should be the easiest way to resolve this issue.

Debugger Console View is not working or GDB is misbehaving

SoftConsole v2021.1 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

Or the following error:
Error: gdb sent 64 bits for a 32-bit register (pc)
```

This indicates a 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 the 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 must 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 triggered 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 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 SoftConsole 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.

The Linux launcher depends on a bash shell, using an alternative shell might cause some paths to be misconfigured and then all depending tools to misbehave.

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`"

Application is stuck in main_first_hart and is not reaching e51()

Depending on the HAL version and project settings this behavior can change. The main_first_hart() in the system_startup.c is declared as a weak. Custom implementation can override the default functionality and the behavior might be different between the projects. When troubleshooting temporary remove the overridden functionality (use the functions shipped with HAL) and double check if all harts are present and not forcefully halted (as some Renode launchers/scripts can halt unused harts). If the issue is resolved, then it might be caused by the HAL v1.5 expecting all harts to be present. A MPFS_HAL_LAST_HART pre-processor define can be used to tweek the new HAL. In case when Renode the launchers/macros/scripts halted hart forcefully. Extra attention is required when:

- porting SoftConsole 6.0 project to later releases
- changing HAL version
- working with a project which provides custom HAL

See: Renode emulation platform

“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.
```

Tab with an error message is opened before breaking into the main function

The following error can be observed (with different addresses):

```
Break at address "0x1000" with no debug information available, or outside of program code.
```

Before loading the elf to the target it might be executing something which is not resolved. Sometimes it is possible to resolve the issue by specifying the sysroot, but this is not reliable as the target might be executing the previous project. Or in case of Renode no project at all.

http://visualgdb.com/gdbreference/commands/set_sysroot

The error message can be suppressed by checking the following checkbox:

```
Window -> Preferences -> C/C++ -> Debug -> Source Not Found -> Only if source file name is
known but not found
```

What are the Robot references in SoftConsole?

SoftConsole v2021.1 is bundled with Nokia’s RED (Robot EDitor) plugin. Robot can be used for automated testing of Renode and real hardware targets. Currently the RED is not supported SoftConsole feature and requires additional manual steps to work (which are not documented yet), however users might try to experiment with it at their own risks.

https://robotframework.org/

https://github.com/nokia/RED

https://renode.readthedocs.io/en/latest/advanced/building_from_sources.html#prerequisites

https://renode.readthedocs.io/en/latest/tutorials/zephyr-ptp-testing.html

“cc1.exe: error: -march=<YOUR_ARCHITECTURE> invalid ISA string”

The ABI and the architecture need to be correct valid combination (rv32 architecture will not work with rv64 ABI), check your project settings. See Packages used section for listed supported multilibs as that might be related issue. Following error indicates the same issue:

“target emulation `elf64-littleriscv' does not match `elf32-littleriscv'”

“can't link hard-float modules with soft-float modules”

When valid, but unsupported multilib combination is selected, then RV64G is used as fallback, this can cause misleading error messages about mixing floats, mixing 32-but/64-bit, but it’s only symptoms of misconfiguring multilibs. The used multilib is compound together from the selections made in the Project Settings -> C/C++ Build -> Settings -> Tool Settings -> Target Processor.

Delete existing build folder (sometimes make clean is not enough), double check if the project settings are correct (if in doubt compare side by side with similar existing working project) and rebuild the project. Many cryptic looking errors can be caused using invalid or unsupported combinations in the project settings.

“cc1.exe: error: requested ABI requires -march to subsume the 'D' extension”

Enabling double floating-point precision (64-bit) on 32-bit architecture targets is not supported. See Packages used section for listed supported multilibs combinations.

“cc1: error: rv32e requires ilp32e ABI”

Valid, but unsupported architecture and ABI combination. All RV32E targets are not supported as listed in the Packages used

“target emulation `elf64-littleriscv' does not match `elf32-littleriscv'”

Similar issue to “cc1.exe: error: -march=<YOUR_ARCHITECTURE> invalid ISA string” when wrong arch/abi combination is selected and the toolchain will use fallback RV64G even on a 32-bit project.

Deeply nested code causes crashes and glitches

Experiencing a CPU abort message inside Renode:

```
[ERROR] u54_1: CPU abort [PC=0x0]: Trying to execute code outside RAM or ROM at
0x0000000000000000.
```

Or causing reboots, glitches or failures (continuous or intermittent) can be a sign of a small stack and a stack overflow, adjust the stack sizes in the linker script or refactor code to conserve the limited stack. See sections:

Static stack profiling

Renode emulation platform

Note: This error can happen on any hart and is not limited to u54_1, nor PolarFireSoC. Stack overflow can be triggered on any platform and thus the error messages or observed behavior might be slightly different.

Renode on startup crashes with an unhandled exception

If the exception message contains the following:

```
        Could not find a part of the path "/<YOUR_PATH>/.renode/history"
```

Then it could have been caused by moving the SoftConsole installation and the config pointing to the old (and non-existent) location of the history file. Correct the path inside the <SC_INSTALL_DIR>/extras/home/.renode/config or just delete the config and let Renode to re-create it.

Error: "Truncated register 16 in remote 'g' packet"

This error message occurs when debugging a Mi-V soft core RISC-V 32-bit target, but the debugger tries to access it as it if was a 64-bit target.

```
Error in final launch sequence
Failed to execute MI command:
-target-select remote localhost:3333
Error message from debugger back end:
Truncated register 16 in remote 'g' packet
Failed to execute MI command:
-target-select remote localhost:3333
Error message from debugger back end:
Truncated register 16 in remote 'g' packet
Truncated register 16 in remote 'g' packet
```

This issue can be resolved in two ways:

1. The recommended way: give the debugger access to the debugging symbols file by going to:

   Debug launch configuration > *Debugger* > *GDB Client Setup* > *Commands*

   and making sure that this command is included where `<proj-name>` represents the project name:

   ```
   file ${config_name:<proj-name>}/<proj-name>.elf
   ```

2. The alternative way: by using an older GDB and explicitly specifying the target RISC-V architecture. Go to

   Debug launch configuration > *Debugger* > *GDB Client Setup* > *Commands*

   and make sure that the following command is present:

   ```
   set architecture riscv:rv32
   ```

   And then, go to:

   Debug launcher > *Debugger* > *GDB Client Setup* > *Executable name*

   and change

   ```
   ${cross_prefix}gdb${cross_suffix}
   ```

   to

   ```
   ${cross_prefix}gdb-8_3${cross_suffix}
   ```

Other useful Documentation

1. Microsemi GitHub: https://github.com/RISCV-on-Microsemi-FPGA
2. RISC-V specifications: https://riscv.org/specifications/
3. Renode's documentation https://renode.readthedocs.io/en/latest/
4. Mi-V Renode webinar series: https://www.microsemi.com/product-directory/fpga-soc/5210-mi-v-embedded-ecosystem#renode-webinar-series

5. 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.
6. 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

2021.04