# SoftConsole v4.0

## Release Notes

# Table of Contents

# SoftConsole v4.0

## Introduction

These are release notes for SoftConsole v4.0.

This document uses `<SoftConsole-install-dir>` as a placeholder for the actual SoftConsole install directory. Where this is mentioned substitute the actual SoftConsole install directory name (e.g. `C:\Microsemi\SoftConsole_v4.0` on Windows or `$HOME/Microsemi/SoftConsole_v4.0` on Linux).

## Overview

### Key Features

- Supports both Windows and Linux.
- Built using the latest industry standard stock free/open source components and tools for ARM® Cortex®-M firmware development.
- Support for SmartFusion® and SmartFusion2 Cortex-M3 and Cortex-M1 firmware development and debugging.
- Uses OpenOCD for debugging and SmartFusion/SmartFusion2 eNVM programming/program download.
- Supports download to and debugging from SmartFusion eSRAM and eNVM, SmartFusion2 eSRAM, eNVM and external RAM (MDDR) and Cortex-M1 RAM.
- Supports FlashPro JTAG programmer for debugging (FlashPro5 on Linux, FlashPro3/4/5 on Windows).
- Supports semi-hosting redirection of standard/file I/O from target board to host debugger.
- Allows users to install arbitrary additional Eclipse plug-ins and features.
- Includes a built-in terminal emulator for connecting to a target board's serial port.
- Provides many new features and added stability compared to SoftConsole v3.4.

### Features not Supported

- Compatibility with SoftConsole v3.4 workspaces/projects/debug launch configurations. SoftConsole v3.4 workspaces/projects/debug launch configurations cannot be used with SoftConsole v4.0. SoftConsole v4.0 workspaces/projects/debug launch configurations cannot be use with SoftConsole v3.4.
- Debugger driven download of programs to non CFI external parallel flash memories.
- Launching Firmware Catalog from SoftConsole.
- Core8051/Core8051s firmware development and debugging. Use Keil C51 Development Tools with Core8051/Core8051s ISD-51 support..

### Quick Start Guide

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

# Installation

## Windows

### Installing

The installer is a 32 bit executable GUI based program named `Microsemi-SoftConsole-v4.0-Windows-Installer.exe`. It must be run with admin privileges. Run the installer and follow the GUI installer wizard instructions on screen.

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

## Linux

Many of the commands below require `root` access. The instructions below use `sudo`. If this does not work then try `su` instead.

### Before Installing

The SoftConsole installer and runtime tools require that certain 32 bit (i686) packages and libraries are installed. Run the following commands to ensure that the required packages are installed.

- `sudo yum install gtk2.i686`
- `sudo yum install libusb1.i686`
- `sudo yum install libXtst.i686`
- `sudo yum install make`
- `sudo yum install ncurses-libs.i686`
- `sudo yum install xdg-utils`

Note:

- On a 32 bit Linux platform omit the ".i686" suffix to the package name. For example for the gtk2 package:

  ```
  sudo yum install gtk2
  ```

- On a 64 bit Linux platform if you get this error

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

  when attempting to install any of these 32 bit packages then first upgrade the relevant 64 bit package and then try again to install the 32 bit package. For example, for the gtk2 package:

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

- If installation of libusb1[.i686] fails then try libusbx[.i686] instead.

- It is recommended that the platform used to run SoftConsole is up to date with all Linux updates.

### Installing

The installer is a 32 bit executable GUI based program named `Microsemi-SoftConsole-v4.0-Linux-x86-Installer`. 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:

```
chmod +x Microsemi-SoftConsole-v4.0-Linux-x86-Installer
```

or else right click on the SoftConsole installer icon in the Linux file manager, choose *Properties*, select the *Permissions* tab and check the *Execute: Allow executing file as program* option and *Close* to save this change.

Run the installer and follow the GUI installer wizard instructions on screen.

If no GUI appears when the installer is run then check that all required packages listed above were installed and refer to the troubleshooting section if necessary.

If the installer still does not run then try running it from the command line with the `--debugconsole` option as this may help identify if there are other dependent packages that are missing:

```
./Microsemi-SoftConsole-v4.0-Linux-x86-Installer –-debugconsole
```

## After Installing

Normally only `root` can access USB devices. To allow users other than `root` to access the FlashPro5 USB device some changes must be made to the system.

1. Install the OpenOCD udev rules file which gives non `root` access to users in the `plugdev` group to the various JTAG and other probes supported by OpenOCD including the FlashPro5 programmer.

   ```
   sudo cp <SoftConsole-install-dir>/openocd/share/openocd/contrib/99-
   openocd.rules /etc/udev/rules.d
   ```

2. The OpenOCD udev rules give non `root` access to all users in the `plugdev` group. So make sure that this group exists and that any user(s) who will use SoftConsole/OpenOCD/FlashPro are members of that group.

   This can be done using the *User Manager* utility which is often accessible from the *System > Administration > Users and Groups* or *Applications > Sundry > Users and Groups* menu options. Alternatively it can be run from the command line using the `system-config-users` command. If the command is not found then the package may be missing, in which case it needs to be installed using `yum install system-config-users`.

   Alternatively the necessary configuration can be carried out on the command line as follows:

   To check if the `plugdev` group already exists run:

   ```
   getent group plugdev
   ```

   If nothing is returned then the `plugdev` group needs to be created:

   ```
   sudo groupadd plugdev
   ```

   Now add to the `plugdev` group those users who will be running SoftConsole:

   ```
   gpasswd -a <username> plugdev
   ```

   Where `<username>` is a placeholder for the actual user name.

3. Reboot the computer and log into the relevant non `root` user account now in the `plugdev` group.

4. Connect the FlashPro5 to a USB port on the computer and check that the FlashPro5 device is recognized by running the `lsusb` command (part of the usbutils package):

   ```
   lsusb –d 1514:
   ```

   If it is then something similar to the following should be displayed:

   ```
   Bus 001 Device 003: ID 1514:2008 Actel
   ```

   If the FlashPro5 is not recognized then review the previous instructions to ensure that they were carried out correctly.

5. Finally check that OpenOCD, run without `root` privileges, can communicate with the target processor via FlashPro5. For SmartFusion and SmartFusion2 boards ensure that `JTAG_SEL` is tied high. Connect the computer to the board via FlashPro5. This example assumes that the target board uses a SmartFusion2 `M2S090` device but adjust the `DEVICE` settings as necessary:

   ```
   cd <SoftConsole-install-dir>/openocd/bin
   export LD_LIBRARY_PATH=`pwd`
   ./openocd –c "set DEVICE M2S090" –f board/microsemi-cortex-m3.cfg
   ```

   The following indicates that communication was successful. If there are any errors then review the previous instructions to ensure that they were carried out correctly.

   ```
   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
   M2S090
   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: S200XTYRZ3
   Info : FlashPro port used: S200XTYRZ3
   Info : clock speed 2000 kHz
   Info : JTAG tap: M2S090.tap tap/device found: 0x1f8071cf (mfg: 0x0e7,
   part: 0xf807, ver: 0x1)
   Info : JTAG tap: M2S090.tap disabled
   Info : JTAG tap: M2S090.dap enabled
   Info : M2S010.cpu: hardware has 6 breakpoints, 4 watchpoints
   ```

## Troubleshooting

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

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

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

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

# Related Microsemi Tools/Resources

## Previous SoftConsole v4.0 Beta/Pre-releases

If a previous beta/pre-release version of SoftConsole v4.0 has been used then it is advisable to recreate any workspaces, projects and debug launch configurations anew.

## Libero SoC/Firmware Catalog

Libero SoC v11.6 or later should be used to create hardware projects and generate/export firmware for use with SoftConsole v4.0.

The Firmware Catalog v11.6 or later can also be used to generate firmware for use with SoftConsole v4.0.

## Firmware Drivers

### CMSIS/HAL Firmware Cores

The following firmware cores (or later versions if available) must be used:

- SmartFusion2 CMSIS Hardware Abstraction Layer 2.3.103
- SmartFusion CMSIS-PAL 2.4.101
- Hardware Abstraction Layer (e.g. for Cortex-M1/DirectCore use) 2.3.101

Make sure that Libero SoC and the Firmware Catalog tools are configured to use this repository and use the above core versions or later versions of available.

**Warning:**

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

### Matching Firmware to the Target Hardware

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

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

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

**Warning:**

- Before importing/copying Libero SoC or Firmware Catalog generated firmware files into a SoftConsole project it is advisable to manually delete all `CMSIS`, `hal`, `drivers` and `drivers_config` folders from the SoftConsole project leaving only the project specific custom source files.
- 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 v4.0 and should not be used.

## FlashPro JTAG Programmer

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

On Windows the FlashPro3/4/5 programmers are supported and the relevant drivers must be installed. On Linux the FlashPro5 programmer only is supported and the post-install configuration steps must have been carried out.

## SoftConsole v3.4

SoftConsole v3.4 workspaces, projects and debug launch configurations are not compatible with SoftConsole v4.0 and should not be used. They will not open or operate correctly. Existing SoftConsole v3.4 workspaces, projects and debug launch configurations must be created anew in SoftConsole v4.0. However this is not an onerous task and is explained elsewhere in the release notes.

Similarly SoftConsole v4.0 workspaces, projects and debug launch configurations are not compatible with SoftConsole v3.4.

# Workspaces

## Example Workspace

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

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

This workspace contains a number of projects and debug launch configurations that are ready to use once the relevant projects have been updated to match the target hardware (e.g. Libero SoC generated `drivers_config` folder copied into the project etc.). It is advisable to copy this workspace before experimenting so that a copy of the original workspace is retained for reference purposes and further copying and experimentation.

### Example Projects

- CM3_GNU_simple_blink: the SmartFusion MSS GPIO Driver firmware core's "Simple Blink" sample project targeting a SmartFusion design with MSS GPIOs [0:4] configured as outputs connected to board LEDs.
- g4m_system_MSS_CM3_app/hw_platform: the SmartFusion2 Starter Kit (SF2-484-STARTER-KIT) SoftConsole "Hello, World" demo app and library project pair from here: http://www.microsemi.com/products/fpga-soc/design-resources/dev-kits/smartfusion2/smartfusion2-starter-kit#documents
- SF2_GNU_GPIO_simple_blink: the SmartFusion2 MSS GPIO Driver firmware core's "Simple Blink" sample project targeting a SmartFusion2 design with MSS GPIOs [0:4] configured as outputs connected to board LEDs.
- Webserver_MSS_CM3_0_app/hw_platform: the SmartFusion Evaluation Kit SoftConsole Webserver demo using FreeRTOS and uIP application and library project pair from here: http://www.microsemi.com/products/fpga-soc/design-resources/dev-kits/smartfusion/smartfusion-evaluation-kit#documents

### 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).

## Creating a New Workspace

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

Note: : Unlike earlier beta/pre-release versions of SoftConsole v4.0 no further manual configuration of the workspace is necessary in order for it to work properly.

# Projects

## Creating a New Project

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

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

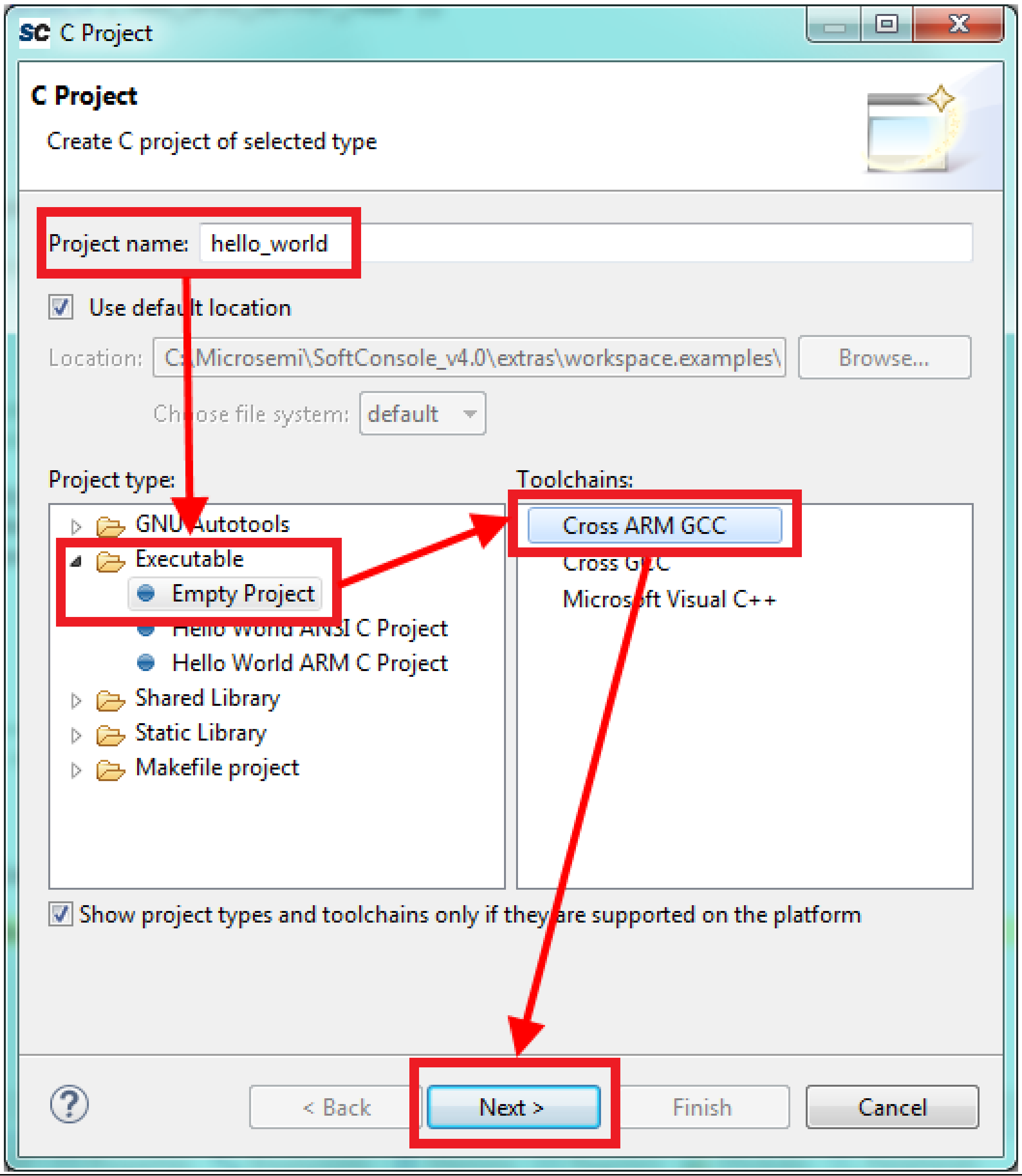
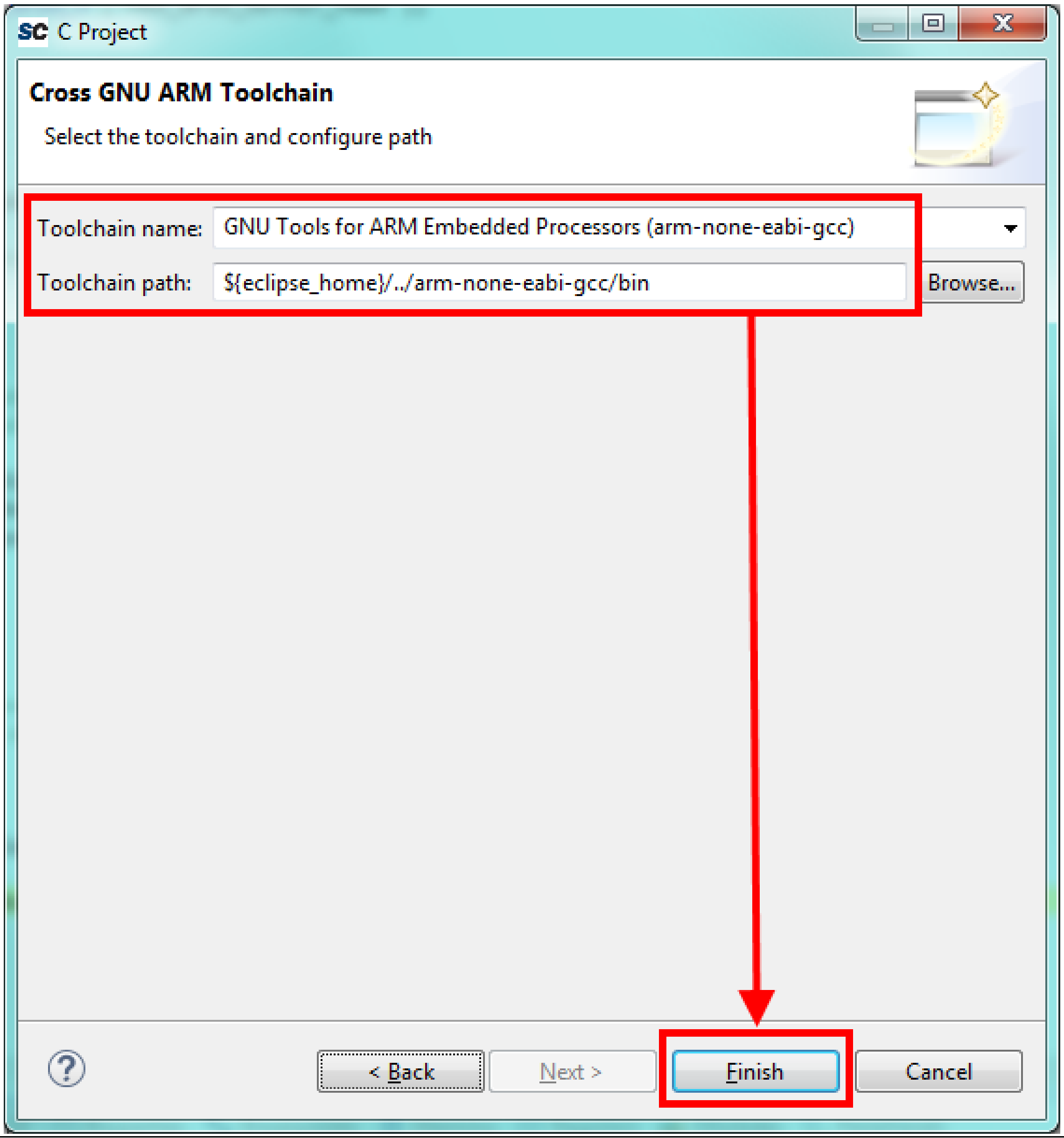

C Project
C Project
Create C project of selected type
Project name: hello_world
Use default location
Location: C:\Microsemi\SoftConsole_v4.0\extras\workspace.examples\
Browse...
Choose file system: default
Project type:
GNU Autotools
Executable
Empty Project
Hello World ANSI C Project
Hello World ARM C Project
Shared Library
Static Library
Makefile project
Toolchains:
Cross ARM GCC
Cross GCC
Microsoft Visual C++
Show project types and toolchains only if they are supported on the platform
< Back
Next >
Finish
Cancel

Figure 1. New Project

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

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


C Project

Cross GNU ARM Toolchain

Select the toolchain and configure path

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

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

**Figure 2. New Project Tool Chain**

## Project Settings

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

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

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

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

## All Targets

### Linker Script

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

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

- SmartFusion2
  `../CMSIS/startup_gcc/debug-in-microsemi-smartfusion2-esram.ld`
- SmartFusion
  `../CMSIS/startup_gcc/debug-in-actel-smartfusion-esram.ld`
- Cortex-M1
  `../hal/CortexM1/GNU/ram-debug.ld`

Note:

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

### Newlib-Nano

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

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

### Create Extended Listing

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

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

### Preprocessor Defines and Includes

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

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

### Optimization Options

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

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

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

### Library Dependencies

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

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

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

### Cross ARM GNU Print Size

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

### Other Options

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

### Specifying Options for All Build Configurations

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

## SmartFusion2 Cortex-M3

### CMSIS

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

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

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

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

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

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

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

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

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

.

### SmartFusion Cortex-M3

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

### Cortex-M1

#### Target Processor

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

#### Linker Options

When using the current Hardware Abstraction Layer (for Cortex-M1/DirectCore) the following linker option must be selected/checked otherwise the project will fail to link:

*Tool Settings* > *Cross ARM C/C++ Linker* > *General* > *Do not use standard start files (-nostartfiles)*

At some point in the future when the Hardware Abstraction Layer is updated to align with the ARM CMSIS standard this option will no longer be required.

## Adding Source Files to Project

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

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

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

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

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

**Warning:** remember that any SoftConsole v3.4 workspaces or projects generated by Libero SoC or the Firmware Catalog cannot be used with SoftConsole v4.0.

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

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

## Building a Project

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

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

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

# Debugging

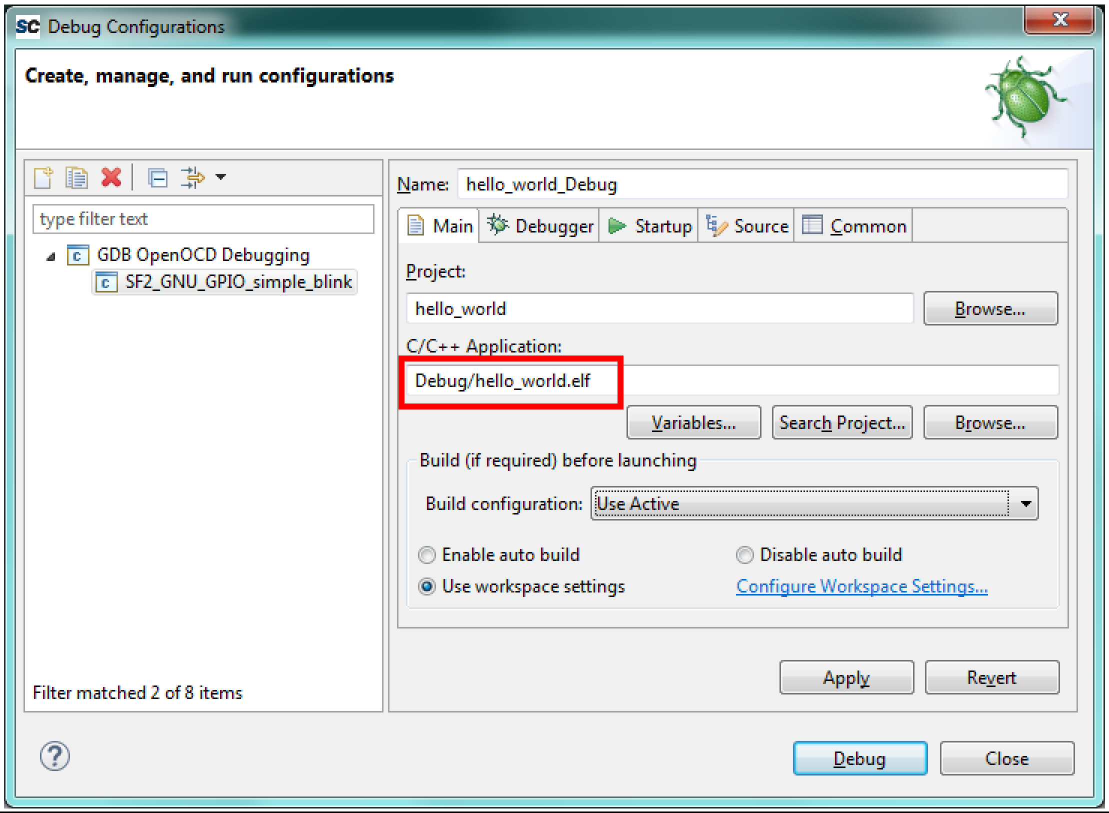
## Debug Launch Configurations

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

1. Select the project in the *Project Explorer* and from the SoftConsole application menu select *Run > Debug Configurations...*

2. In the *Debug Configurations* dialog select *GDB OpenOCD Debugging* and click on the *New launch configuration* button which will create a new debug launch configuration for the previously selected project.

3. On the *Main* tab ensure that the *C/C++ Application* field contains the correct executable name. Note that using forward slashes in paths here aids portability of projects and debug launch configurations between Windows and Linux:


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: hello_world_Debug

Main | Debugger | Startup | Source | Common

Project: hello_world

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

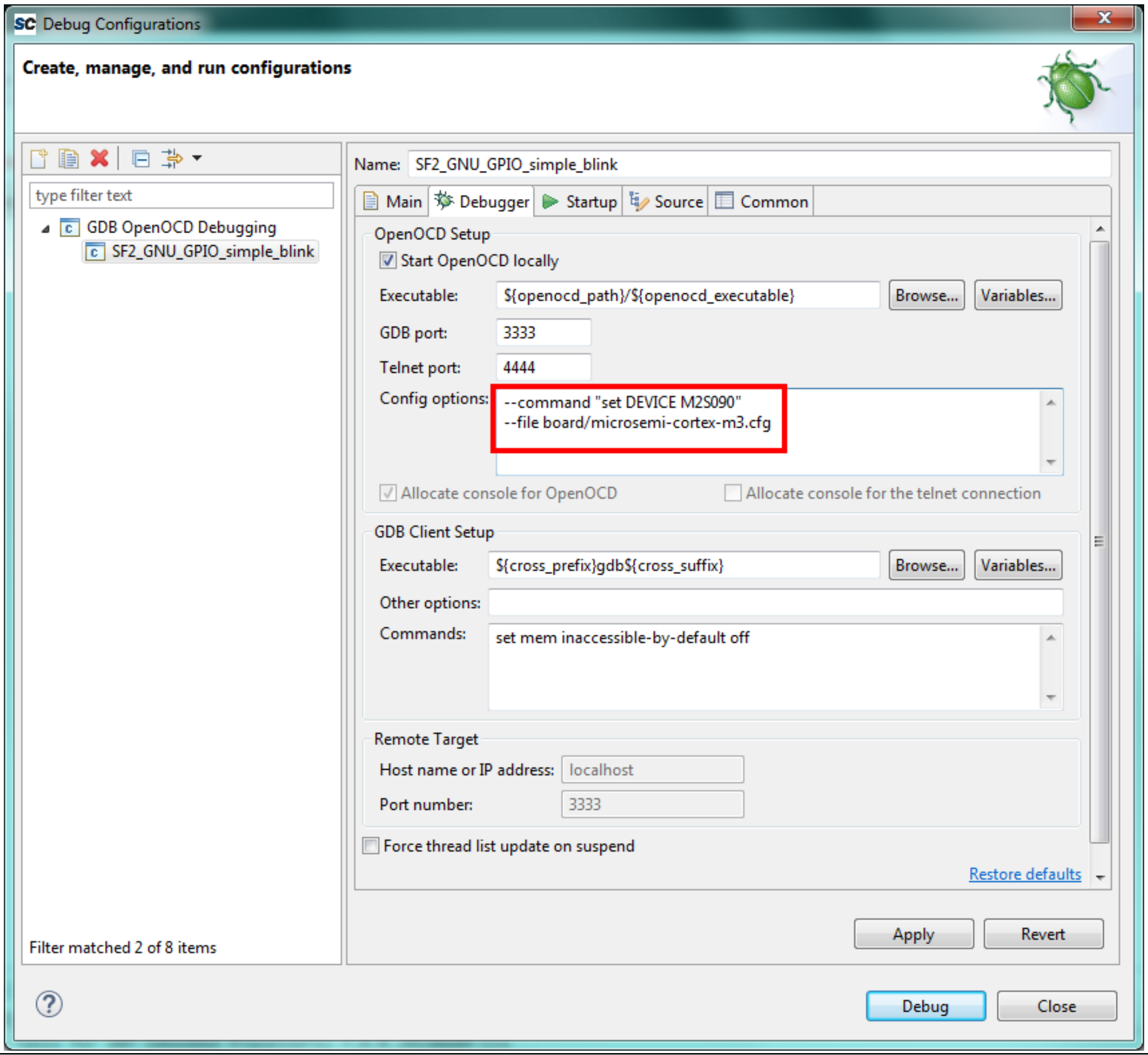
Build (if required) before launching

Build configuration: Use Active

( ) Enable auto build
( ) Disable auto build
(●) Use workspace settings

**Figure 3. Debug Launch Main**

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



Name: SF2_GNU_GPIO_simple_blink

OpenOCD Setup
☑ Start OpenOCD locally
Executable: ${openocd_path}/${openocd_executable}
GDB port: 3333
Telnet port: 4444
Config options:

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

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

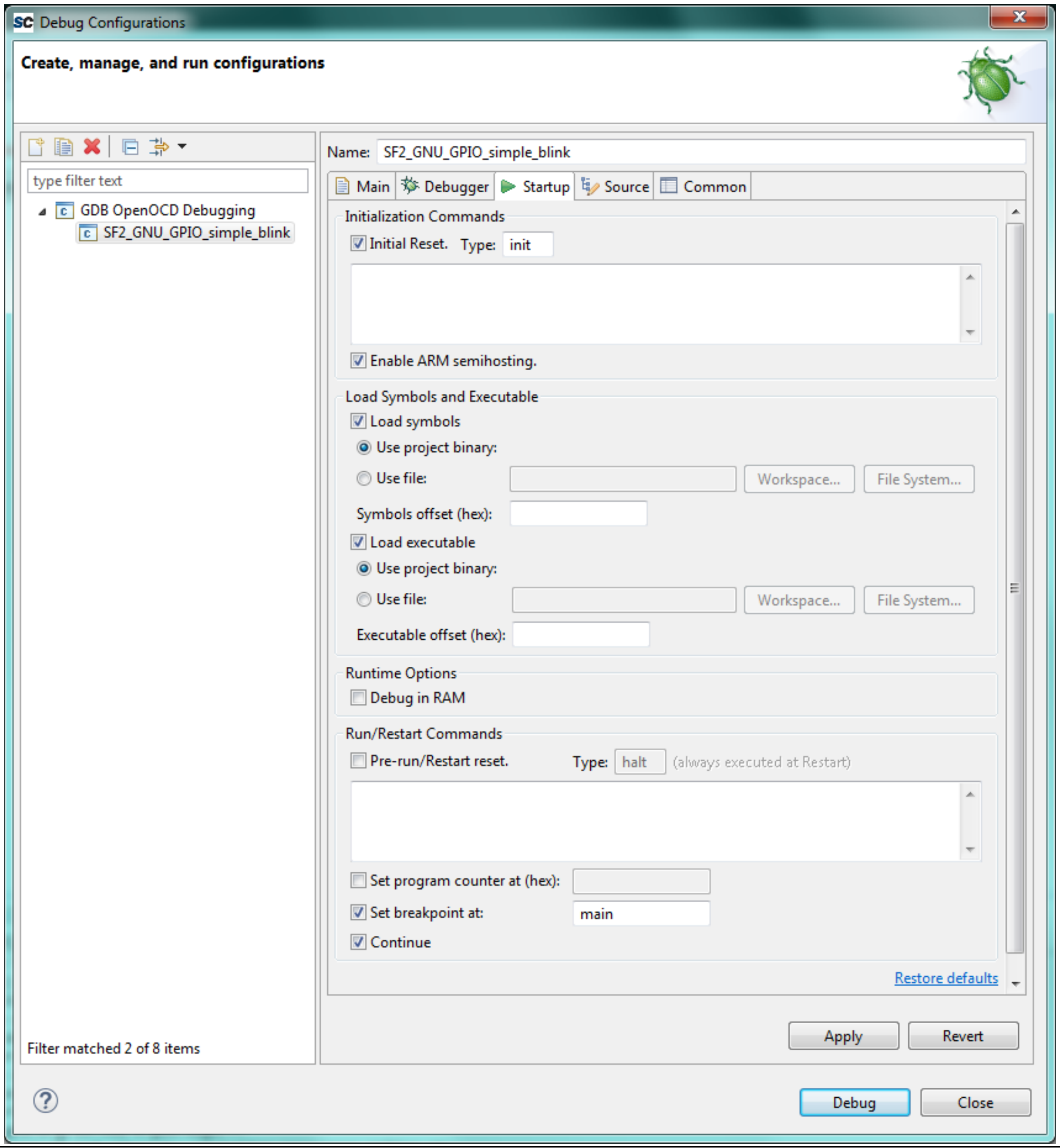
GDB Client Setup
Executable: ${cross_prefix}gdb${cross_suffix}
Other options:
Commands: set mem inaccessible-by-default off

Remote Target
Host name or IP address: localhost
Port number: 3333

☐ Force thread list update on suspend

**Figure 4. Debug Launch Debugger**

5. On the *Startup* tab the default settings should be configured as shown below and these are the default settings so do not change them unless absolutely necessary and you understand what effect these changes will have.
*Initialization Commands > Initial Reset* must be checked and *Type* set to *init*. *Enable ARM semihosting* can be enabled whether or not semihosting will be used. *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.


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 Startup

6. 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).

`

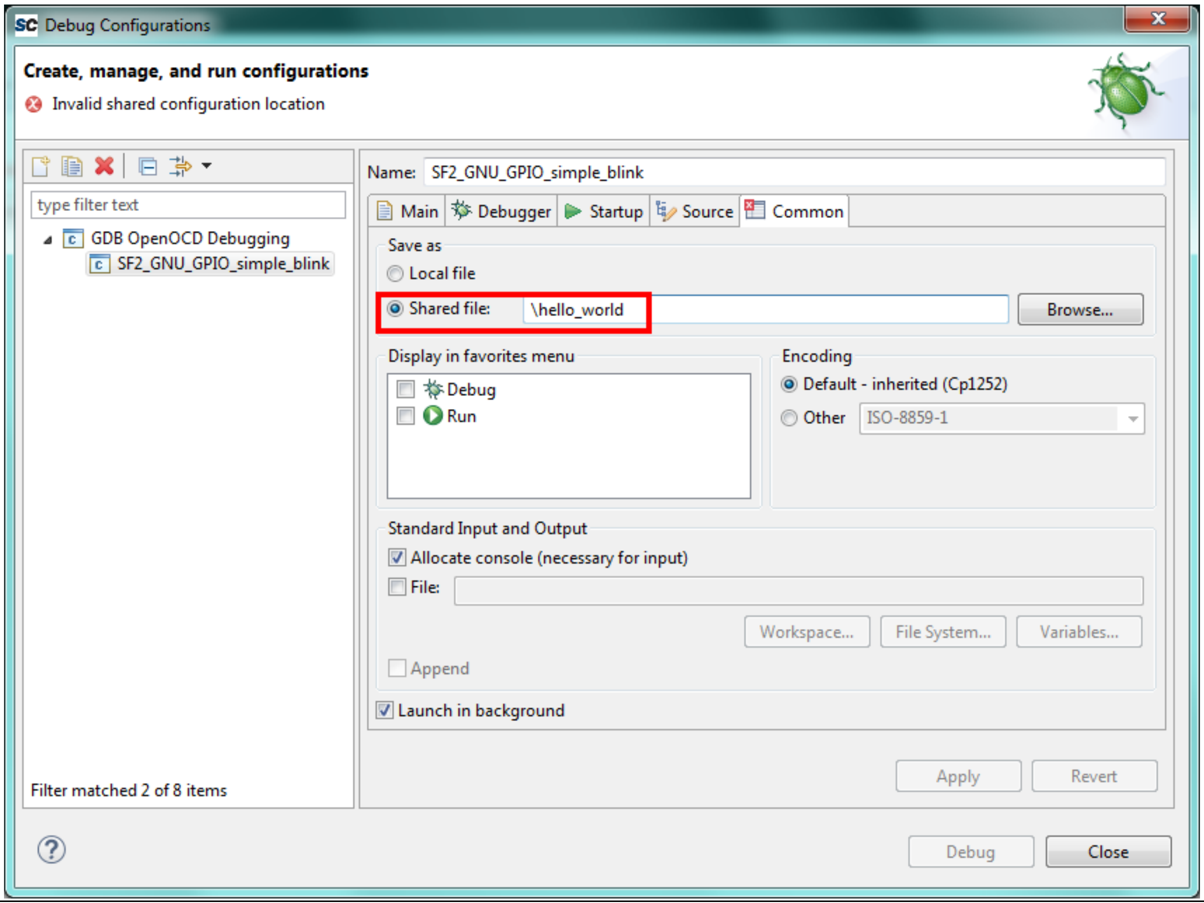

Debug Configurations
Create, manage, and run configurations
Invalid shared configuration location
type filter text
GDB OpenOCD Debugging
SF2_GNU_GPIO_simple_blink
Filter matched 2 of 8 items
Name: SF2_GNU_GPIO_simple_blink
Main
Debugger
Startup
Source
Common
Save as
Local file
Shared file: \hello_world
Browse...
Display in favorites menu
Debug
Run
Encoding
Default - inherited (Cp1252)
Other ISO-8859-1
Standard Input and Output
Allocate console (necessary for input)
File:
Workspace...
File System...
Variables...
Append
Launch in background
Apply
Revert
Debug
Close

**Figure 6. Debug Launch Common**

## OpenOCD Command Line Options or 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_SPEED 1000"
```

or together separated by semi-colons

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

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

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

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

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

## Cortex-M1 Board Script

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

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

Because Cortex-M1 is a soft processor implemented in FPGA fabric the structure of the target system on chip is not known in advance – e.g. location and size of RAM, flash memories etc. When targeting a Cortex-M1 to download to/debug from RAM the base address and size of RAM must be specified – e.g.:

```
--command "set RAM_BASE 0x00000000; set RAM_SIZE 0x7000"
--file board/microsemi-cortex-m1.cfg
```

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

**Warning:** at the moment support for non CFI flash memory is not provided in SoftConsole.

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 USE_UTAG N"
```

### FlashPro JTAG Speed

The SoftConsole OpenOCD scripts use a default JTAG clock speed of 2MHz. 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_SPEED 1000"
```

or

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

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

### Other OpenOCD Options

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

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

## Board Configuration for FlashPro Debugging

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

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

## Using a Debug Session

### Launching a Debug Session

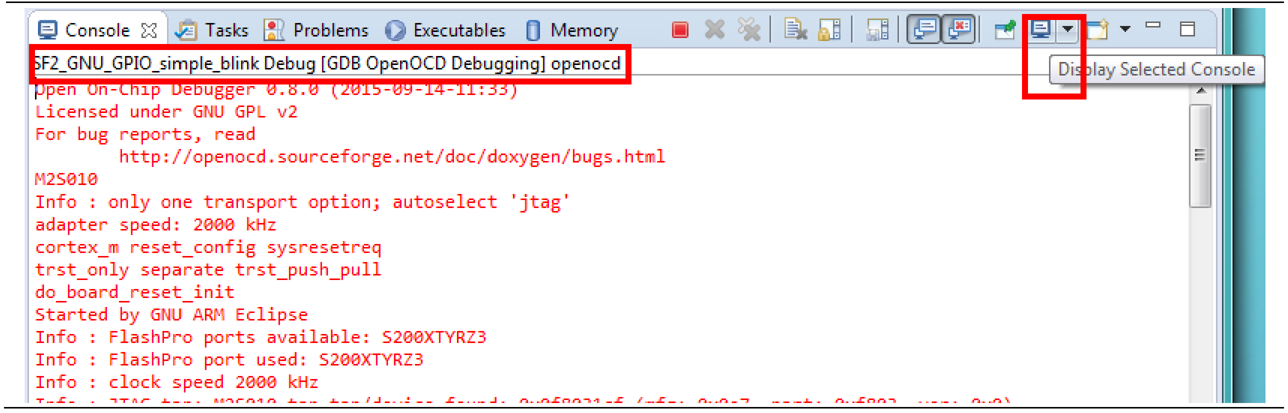
Select the project in the *Project Explorer*, right click on it and from the context menu select *Debug As* > *Debug Configurations*, select the relevant debug launch configuration and click *Debug*.

### Memory Monitor

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

### Console View

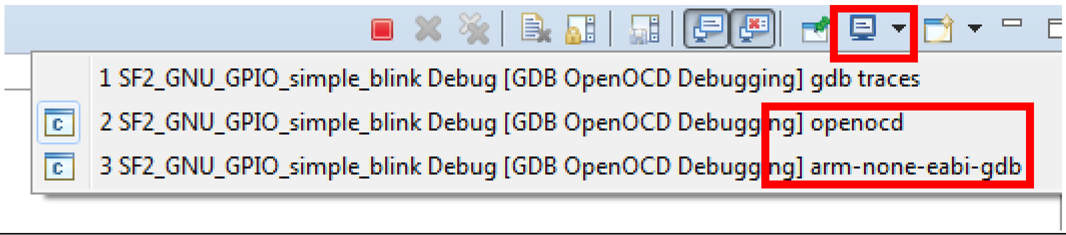
During a debug session SoftConsole can display a number of different consoles in the *Console* view. By default the OpenOCD console is displayed showing OpenOCD output:



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

**Figure 7. Debug Console Openocd**

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 Select Console**

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

## Built-in Serial Terminal View

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

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

## Debug Using a Specific FlashPro Programmer

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

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

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

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

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

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

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

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

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

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

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

## Debugging Using a Non FlashPro JTAG Interface

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

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

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

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

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

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

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

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

1. Copy `.../board/microsemi-cortex-m3.cfg` to `.../board/microsemi-cortex-m3-olimex.cfg`

2. In `.../board/microsemi-cortex-m3-olimex.cfg` replace

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

   with

   ```
   # Olimex ARM-USB-TINY-H
   source [find interface/ftdi/olimex-arm-usb-tiny-h.cfg]
   ```

3. In the debug launch configuration put the following:

   ```
   --command "set DEVCE M2S090; set JTAG_SEL L"
   --file board/microsemi-cortex-m3-olimex.cfg
   ```

4. 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).

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

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

### How to Connect to/Debug a Running Program

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

1. The program/project built must match the program running on the target – i.e. exactly the same code, linker script etc.
2. On the *Startup* page of the debug launch configuration...
3. Clear the *Initial Reset* checkbox
4. In the *Initialization Commands* text field enter `monitor halt`
5. Clear the *Load Symbols and Executable* > *Load Executable* checkbox

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

TODO: screenshot.

## Troubleshooting

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

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

# Other Features

## Semihosting

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

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

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

```
#include <stdio.h>

extern void  initialise_monitor_handles(void);

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

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

## Integer Only Newlib Support

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

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

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

## Static Stack Profiling

GCC supports static stack usage analysis/profiling.

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

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

# Known Issues

## Initial startup may be slow

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

## Flash Programming

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

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

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

## Build Project Context Menu Option Sometimes Disabled

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

## Windows Firewall and OpenOCD

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

## Multiple Debug Sessions

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

## Memory Monitor Fails to Display

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

## Other useful Documentation

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

# Free/Open Source Packages

## Packages Used

Microsemi SoftConsole v4.0 uses a number of free and/or open source packages.

.

| Package | Version |
|---|---|
| Oracle Java SE | 8u60 |
| Eclipse/CDT | Eclipse 4.4.2 (Luna SR2)<br>CDT 8.6.0 for Eclipse Luna with Microsemi modified starter.exe (on Windows) to allow for graceful termination of OpenOCD. |
| GNU ARM Eclipse plugins | 2.9.3-201508190739 |
| GCC ARM Embedded (GNU Tools for ARM Embedded Processors) | 4.9-2015-q2-update |
| ARM Cortex Microcontroller Software Interface Standard (CMSIS) | V4.3 |
| OpenOCD | 0.8.0 with Microsemi enhancements |
| GNU ARM Eclipse Build Tools (Windows only) | 2.6-201507152002 includes<br>GNU make 4.1 and BusyBox 1.24.0-git |
| Inno Setup (Windows only) | 5.5.6 |
| InstallJammer (Linux only) | 1.2.15 |
| RXTX Java library and Eclipse plug-ins | 2.1-7r4 |

## Licensing

Microsemi SoftConsole v4.0 uses a number of free and/or open source packages whose use is governed by the specified license agreements.

| Package | License |
|---|---|
| Oracle Java SE | Oracle Binary Code License Agreement for the Java SE Platform Products and JavaFX |
| Eclipse/CDT | Eclipse Public License |
| GNU ARM Eclipse plugins | Eclipse Public License |
| GCC ARM Embedded (GNU Tools for ARM Embedded Processors) | Various – see here |
| ARM Cortex Microcontroller Software Interface Standard (CMSIS) | End user licence agreement for the Cortex Microcontroller Software Interface Standard (CMSIS) deliverables |
| OpenOCD | GNU General Public License v3 |
| GNU ARM Eclipse Build Tools (Windows only)<br>GNU make 4.1 and BusyBox 1.23 | GNU make GNU General Public License v3<br>BusyBox GNU General Public License v3 |
| Inno Setup | Inno Setup license |
| InstallJammer | GNU General Public License with exception |
| RXTX Java library and Eclipse plug-ins | LGPL v2.1 + Linking Over Controlled Interface |

## Documentation

| Package | Documentation |
|---|---|
| Eclipse/CDT | Eclipse documentation<br>CDT documentation |
| GNU ARM Eclipse plugins | Documentation |
| GCC ARM Embedded (GNU Tools for ARM Embedded Processors) | Release.txt listing the specific versions of tools used<br>GCC 4.8 Manual<br>GDB documentation<br>GNU binutils documentation<br>Newlib documentation |
| ARM Cortex Microcontroller Software Interface Standard (CMSIS) | Documentation |
| OpenOCD | User's Guide |
| RXTX Java library and Eclipse plug-ins | Documentation |

## Supported Platforms

- Operating systems
  - Windows
    - Windows 7 Professional SP1 (32 and 64 bit versions)
    - Windows 8.1 Professional (32 bit and 64 bit versions)
    - Windows 10 Professional (32 bit and 64 bit versions)
  - Linux
    - CentOS 6.7 (32 and 64 bit versions)
    - CentOS 7.1-1503 (CentOS 7 onwards is 64 bit only)
    - Note: CentOS 5.11 (32 and 64 bit versions) is not currently supported
- Boards
  - SmartFusion2
    - SmartFusion2 Advanced Development Kit – M2S150-ADV-DEV-KIT
    - SmartFusion2 Security Evaluation Kit – M2S090TS-EVAL-KIT
    - SmartFusion2 Starter Kits – SF2-STARTER-KIT and SF2-484-STARTER-KIT
  - SmartFusion
    - SmartFusion Evaluation Kit – A2F-EVAL-KIT-2
    - SmartFusion Development Kit – A2F500-DEV-KIT-2
  - Cortex-M1
    - Fusion Embedded Development Kit – M2AFS-EMBEDDED-KIT-2
- Other hardware
  - FlashPro3/4/5 on Windows
  - FlashPro5 on Linux
- Other software
  - Libero
    - Libero SoC v11.6
    - Firmware Catalog v11.6
  - Firmware
    - SmartFusion2 CMSIS Hardware Abstraction Layer 2.3.103
    - SmartFusion CMSIS-PAL 2.4.101
    - Hardware Abstraction Layer 2.3.101 (for Cortex-M1 and DirectCore peripheral driver support)

# Product Support

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

## Customer Service

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

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

## Customer Technical Support Center

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

## Technical Support

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

## Website

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

## Contacting the Customer Technical Support Center

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

### Email

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

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

### My Cases

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

### Outside the U.S.

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

## ITAR Technical Support

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

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

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

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

**E-mail**: sales.support@microsemi.com

51300135-1/09.15