SoftConsole v3.4

User's Guide

Table of Contents

Introduction

Prerequisite knowledge

It is assumed that the reader has some familiarity with FPGA and embedded C firmware development flows and tools.

Overview

SoftConsole is Microsemi's free software development toolchain that enables the rapid development of C, C++ and assembler based embedded firmware for the full range of Microsemi CPUs.

Key features of SoftConsole are:

- Provides a flexible and intuitive Eclipse based GUI IDE which integrates and coordinates all of the underlying development tools and facilitates the creation, management, building and debugging of embedded firmware projects.
- Bundles all development tools required to create embedded firmware using C, C++ and assembler – including make, compiler, assembler, linker, debugger and ancillary development tools.
- The Eclipse based GUI IDE provides a front end to the GDB debugger for interactive debugging of program code on a hardware target connected via FlashPro programmer device.
- Supports the full embedded firmware development lifecycle from original creation of firmware projects, thru iterative and interactive debugging of debug builds on target hardware thru to the production of the ultimate release build executable suitable for storage in and execution from non volatile memory.
- Supports (embedded and external) flash programming of program images for interactive debugging and production execution purposes.
- Integrates with other Microsemi tools such as Libero and Firmware Catalog for ease of use and parallel development of hardware and firmware.

Supported platforms

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

Supported CPUs

SoftConsole supports the full range of Microsemi CPUs. Refer to the web links for more detailed information and documentation relating to each CPU.

- Cortex-M3 (http://www.microsemi.com/soc/products/mpu/cortexm3) which is implemented as part of the Microcontroller Subsystem (MSS) in the following devices
    - SmartFusion2 SoC (http://www.microsemi.com/soc/products/smartfusion)
    - SmartFusion cSoC (configurable System on Chip) (http://www.microsemi.com/soc/products/smartfusion2)
- Cortex-M1 (http://www.microsemi.com/soc/products/mpu/cortexm1)

- CoreMP7 ARM7TDMI-S (http://www.microsemi.com/soc/products/mpu/coremp7)
- Core8051s (http://www.microsemi.com/soc/products/ip/search/results.aspx?n=8051&fn=none&pv=none&mk=none)

Installing

Recent releases of Microsemi Libero SoC bundle SoftConsole so that it can be installed when installing Libero and other associated tools.

Alternatively to install SoftConsole on a standalone basis download the install image from the link below, run the installer and click through the installation wizard:

http://www.microsemi.com/soc/download/software/softconsole/default.aspx

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

In order to use SoftConsole to download and debug programs on a hardware target (for example, a development board), a FlashPro programmer device must be connected to a USB port on the computer the required drivers installed. This should be done before attempting to download and

debug programs on a hardware target. Failing to attach the required programmer results in the following error when attempting to access the hardware target:

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

Refer to the FlashPro installation instructions for more information about installing the required FlashPro software, hardware, and drivers.

http://www.microsemi.com/soc/download/program_debug/flashpro/default.aspx

SoftConsole supports FlashPro4, FlashPro3 (including LCPS and on-board FlashPro3 programmers built onto certain development boards) and FlashPro Lite (for 8051 only).

http://www.microsemi.com/soc/products/hardware/program_debug/flashpro/default.aspx

SoftConsole can be uninstalled using the standard Windows **Add/Remove Programs** utility, using the **Start** menu entry for the uninstaller (**Start > All Programs > Microsemi SoftConsole > Uninstall Microsemi SoftConsole IDE**) or using the Libero uninstaller if it was originally installed using the Libero installer.

Constituent components

SoftConsole comprises a number of constituent components that are described briefly here. For more detailed information and documentation please refer to each component's specific home and documentation pages. For details of the specific version of each component used please refer to the SoftConsole release notes.

Java

Eclipse, CDT and related plug-ins are Java based so SoftConsole bundles a copy of the Oracle Java Standard Edition runtime.

Eclipse

The Eclipse Java based platform and workbench provide the main user interface to SoftConsole allowing for the creation and management of workspaces and (in conjunction with the Eclipse CDT and related plugins) embedded firmware C/C++ projects.

Generic documentation about Eclipse is available from here: http://help.eclipse.org/galileo/index.jsp?nav=/0[1]. This documentation explains the general concepts, structure, function and appearance of the Eclipse platform and workbench.

The Eclipse home page is here: http://www.eclipse.org

Eclipse CDT

The Eclipse CDT (C/C++ Development Tooling) adapts the generic Eclipse development environment and provides specific support for C /C++ development. To quote the Eclipse CDT home page:

"The CDT Project provides a fully functional C and C++ Integrated Development Environment based on the Eclipse platform. Features include: support for project creation and managed build for various toolchains, standard make build, source navigation, various source knowledge tools, such as type hierarchy, call graph, include browser, macro definition browser, code editor with syntax highlighting, folding and hyperlink navigation, source code refactoring and code generation, visual debugging tools, including memory, registers, and disassembly viewers."

The version of the Eclipse CDT bundled with SoftConsole is further modified and enhanced through a number of additional Eclipse plug-ins in order to provide specific configuration, development and debugging support for Microsemi CPU targets.

Taken together the Eclipse platform and workbench, Eclipse CDT and additional related plug-ins provide the framework for the management and integration of all underlying development and debugging tools and also provide the main SoftConsole IDE front end visible to the end user for the creation, management, configuration, development and debugging of firmware projects.

[1] *SoftConsole v3.4 bundles Eclipse Galileo 3.5*

Generic documentation about the Eclipse CDT is available from here: http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.cdt.doc.user/concepts/cdt_o_home.htm. This documentation explains the general concepts, structure, function and appearance of the CDT.

The Eclipse CDT home page is here: http://www.eclipse.org/cdt

CodeSourcery Sourcery G++ Lite – GNU Toolchain for ARM Processors[2]

This component provides a complete set of development tools for ARM CPUs, in particular the Microsemi Cortex-M3, Cortex-M1 and CoreMP7 ARM7TDMI-S.

- GCC (The GNU Compiler Collection) C and C++ compilers
- GDB (The GNU Project Debugger) with Microsemi modifications for ARM CPU hardware target debugging
- GNU Binutils (Binary Utilities) including assembler and linker
- GNU Make
- newlib C standard library

More detailed information and documentation on each of these tools are available from the relevant home page and documentation links. For example the GCC and GNU Binutils ld linker documentation explains all of the command line options supported by these tools some of which might be needed in order to specify them in a particular SoftConsole project using the Eclipse/CDT project settings.

- GCC (The GNU Compiler Collection) C and C++ compilers
  - Home page: http://gcc.gnu.org
  - Documentation: http://gcc.gnu.org/onlinedocs
- GDB (The GNU Project Debugger)
  - Home page: http://www.gnu.org/software/gdb
  - Documentation: http://www.gnu.org/software/gdb/documentation
- GNU Binutils (Binary Utilities) including assembler and linker
  - Home page: http://www.gnu.org/software/binutils
  - Documentation: http://sourceware.org/binutils/docs-2.23.1
- GNU Make
  - Home page: http://www.gnu.org/software/make
  - Documentation: http://www.gnu.org/software/make/manual
- newlib C standard library
  - Home page: http://sourceware.org/newlib
  - Documentation: http://sourceware.org/newlib/docs.html

8051 development tools

Core8051s software development and debugging is provided by way of the following tools:

[2] CodeSourcery Sourcery G++ Lite is now owned by Mentor Graphics and branded Sourcery CodeBench Lite. See here: http://www.mentor.com/embedded-software/codesourcery.

- SDCC Small Device C Compiler[3]
    - Home page: http://sdcc.sourceforge.net
    - Documentation: http://sdcc.sourceforge.net/doc/sdccman.pdf
- CodeSourcery omf2elf OMF to ELF executable file format converter utility
- GDB (The GNU Project Debugger) with Microsemi modifications for 8051 CPU hardware target debugging
- GNU Binutils (Binary Utilities) with Microsemi modifications for 8051 CPU support

Hardware target debug support tools

The following tools sit between GDB and the relevant target CPU in order to allow SoftConsole to debug programs that are running on a hardware target. Debug connectivity is via a FlashPro programmer device. These tools translate between GDB Remote Serial Protocol and CPU specific JTAG debug commands and responses.

- Cortex-M1/Cortex-M3: CoreSourcery ARM Debug Sprite with Microsemi modifications and flash programming support
- CoreMP7 ARM7TDMI-S: FS2 In-Target Analyzer for ARM Processor Cores
- Core8051s: C8051 Debug Sprite with Microsemi modifications and flash programming support

[3] In compliance with the relevant licenses governing the use of SDCC source code is available on request from Microsemi.

Related Microsemi tools and resources

When using SoftConsole to develop embedded firmware for Microsemi CPU targets the following Microsemi tools may also be relevant.

Firmware cores

Firmware cores are pre-packaged bundles of embedded firmware header and source files, documentation and fully self contained sample projects supporting specific embedded firmware development tools (including SoftConsole), CPU targets, DirectCores and SmartFusion/SmartFusion2 MSS (Microcontroller Subsystem) peripherals and services.

Firmware cores are published through the Microsemi firmware repository and the Firmware Catalog and Libero provide support for locating, browsing, downloading and generating firmware core drivers, sample projects and documentation. Firmware cores facilitate and accelerate embedded firmware development by obviating the need to write low level code that interacts with the raw hardware platform.

The sample projects bundled with firmware cores are a useful starting point getting familiar with SoftConsole and as a reference point and basis for the creation of custom application projects.

When targeting SmartFusion or SmartFusion2 Cortex-M3 designs it is important to remember to update sample projects with the `<libero-project-root>/firmware/drivers_config` folder and contents generated by Libero for the target design in order to ensure that the firmware project matches the target hardware. Failure to do so may result in unexpected behaviour when executing the firmware.

Firmware cores fall into two main categories

- Hardware abstraction layers
  - Hardware Abstraction Layer underlying the DirectCore peripheral drivers
  - SmartFusion CMSIS-PAL
  - SmartFusion2 CMSIS Hardware Abstraction Layer
- Drivers
  - Firmware drivers for FPGA fabric based DirectCore peripherals
  - Firmware drivers for SmartFusion MSS peripherals
  - Firmware drivers for SmartFusion2 MSS peripherals and system services

The hardware abstraction layer firmware cores provide

- a base platform on which other peripheral drivers sit
- C startup code that runs before `main()` is called and which carries out initialization of the hardware and runtime firmware platforms
- useful example linker scripts which can be reused as-is or adapted to the needs of a specific target. These include linker scripts for a variety of memory map configurations, interactive download/debug and creation of “production” firmware intended for storage in and booting from embedded or external flash memory.[4]

[4] *Linker scripts for downloading to and debugging from a specific target memory are generally named* `debug-in-...` *or* `...-debug` *while those for producing production firmware images for storage in and*

Refer to the Firmware Catalog home page, documentation and online help for more information about the Firmware Catalog and the firmware cores to which it provides access. Detailed documentation about each individual firmware core can also be accessed using the Firmware Catalog.

- Home page: http://www.microsemi.com/soc/products/software/firmwarecat
- Documentation: http://www.microsemi.com/soc/products/software/firmwarecat/default.aspx#docs

Libero

Libero is used to create hardware designs for Microsemi FPGAs. SoftConsole can be used to develop embedded firmware for Microsemi CPU based systems created using Libero.

Libero SoC can integrate with the embedded firmware development toolchain (including SoftConsole, IAR Embedded Workbench and Keil MDK-ARM) allowing for closer co-development of the embedded hardware and firmware. Libero can create the required workspace/project files and embedded firmware drivers and related files matching the target platform and the embedded firmware tool (e.g. SoftConsole) can be launched directly from Libero SoC to work on these. A prerequisite for this to happen is that the embedded firmware toolchain is configured and selected in Libero under **Tool Profiles**.

Using Libero in this way to integrate with the embedded firmware development toolchain and generate the required workspace, project and firmware core files ensures that the firmware closely matches the target hardware in terms of the memory map and drivers used.

When Libero is used in conjunction with SoftConsole in this way the SoftConsole workspace is generated into `<libero-project-root>/SoftConsole`.

Whether or not the Libero integration with the embedded firmware toolchain is used Libero generates firmware related files into `<libero-project-root>/firmware`.

If SoftConsole is used standalone to target SmartFusion or SmartFusion2 Cortex-M3 hardware designs then it is critical that firmware files generated by Libero into `<libero-project-root>/firmware/drivers_config` are copied into the SoftConsole project in order for the firmware to correctly match the target hardware design. Failure to do so may result in unexpected behaviour when executing the firmware.

Refer to the Libero home page, documentation and online help for more information about Libero's support for embedded firmware development.

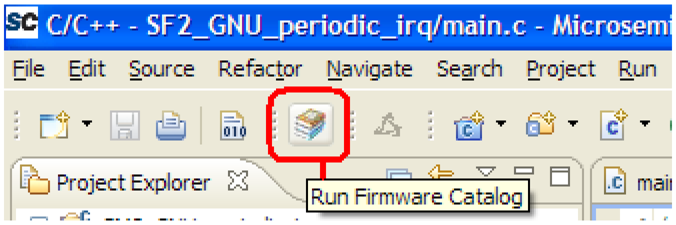
- Home page: http://www.microsemi.com/soc/products/software/libero
- Documentation: http://www.microsemi.com/soc/products/software/libero/docs.aspx

Firmware Catalog

The Firmware Catalog supports the locating (in a remote repository), downloading (from remote repository to local vault) and generation (from the local vault into a local folder or SoftConsole project) of firmware driver packages and sample projects. It can be run standalone or else from within SoftConsole when the **C/C++** perspective is active using the **Run > Run Firmware Catalog** menu option or the **Run Firmware Catalog** toolbar button:

---

booting (and optional relocation) from embedded or external flash memory are generally named `production-...`, `boot-from-...` *or* `run-from-...`


C/C++ - SF2_GNU_periodic_irq/main.c - Microsemi
File Edit Source Refactor Navigate Search Project Run
Project Explorer
Run Firmware Catalog
main

To generate firmware drivers into a project ensure that the project is selected/highlighted in SoftConsole's **Project Explorer** view before launching and using the Firmware Catalog. If no project is selected then the driver will be generated in a folder named `default_fc_folder` in the current workspace.

When a sample project bundled with a firmware driver core is generated the resulting project must be imported into the SoftConsole workspace before it appears in the **Project Explorer** view and can be used.

The Firmware Catalog can be used to update or add to the firmware drivers used by an existing project – including a hardware platform library project created by Libero. It can also be used to generate sample projects that can be used as a starting point for creating a new application.

Refer to the Firmware Catalog home page, documentation and online help for more information about the services offered by the Firmware Catalog and how to use them.

- Home page: http://www.microsemi.com/soc/products/software/firmwarecat
- Documentation: http://www.microsemi.com/soc/products/software/firmwarecat/default.aspx#docs

FlashPro

When debugging SoftConsole connects to the hardware target and communicates with the CPU target using a FlashPro programmer. The FlashPro software provides the drivers needed by SoftConsole to communicate with the FlashPro programmer.

The FlashPro software can also be used to program production firmware produced by SoftConsole into Fusion, SmartFusion or SmartFusion2 ENVM embedded flash by way of a suitable ENVM data storage client defined in the design and associated programming file and associated with the production firmware executable image.

- Home page: http://www.microsemi.com/soc/products/hardware/program_debug/flashpro
- Documentation: http://www.microsemi.com/soc/products/hardware/program_debug/flashpro/default.aspx#docs

Suggested firmware development flow

1. Create the target hardware design using Libero and related tools.
2. Program the hardware design to a development board.
3. Configure SoftConsole as the Libero **Software IDE** in **Tool Profiles** and allow Libero to drive the generation and updating of the SoftConsole workspace, application/hardware platform library projects and firmware cores and then launch SoftConsole from Libero.
4. Note that by convention projects are often and created and configured to provide two build configurations – a debug and a release configuration. The debug configuration is normally not optimized and includes full debug symbolic information for ease of interactive debugging. The release configuration is normally optimized which, even if full debug symbolic information is included, usually makes interactive debugging difficult (e.g. line tracking in the debugger may not work). All Microsemi projects (e.g. firmware core example projects, Libero created application projects, other demo, sample and reference designs) following this convention. However this is just a convention and a project can be configured to have any number and type of build configurations as needed.

5. If necessary tune the project settings (e.g. compiler/assembler/linker options, linker scripts etc.) to the specific needs of the target system. When doing this it may be necessary to refer to the documentation about the relevant underlying tools (e.g. GCC C/C++ compiler, GNU assembler, GNU ld linker, GNU Binutils, SDCC etc.) for information about tool options and switches that can be entered into the SoftConsole project properties.
6. A key consideration at this stage is matching the firmware project's knowledge of the target system's memory spaces and memory map with that of the actual target system. Libero provides some assistance with this by generating a memory map for a CPU based system and even skeleton projects. It is important that the firmware access memory regions and memory mapped peripherals using the appropriate addresses. To this end an appropriate linker script or equivalent specification of the target memory is required. Often the CMSIS example linker scripts provide a useful starting point for creating or adapting a linker script suitable for a specific firmware project and target hardware system.
7. Having built the program using a suitable linker script create a suitable debug launch configuration in SoftConsole and iteratively and interactively download and debug the program on the target platform until it is working satisfactorily. By convention projects usually have a debug build configuration suitable for this purpose.
8. Once the debug build configuration is working build a release confguration (e.g. including optimization) and ensure that this executes satisfactorily. If it does not then further debugging using the debug configuration may be required. Debugging of optimized builds can be difficult due to the lack of reliable source line tracking etc.
9. When the release configuration firmware is executing satisfactorily the next step is to create a production firmware image. This is a normally an optimized/release configuration build linked in such a way that it is suitable for storage in flash memory so that it can boot on power up, optionally relocate in part or full to RAM and then continue executing as part of the embedded system. The CMSIS/HAL firmware core production linker scripts[5] may be suitable (as-is or adapted) for creating a production firmware image for storage in and boot time execution from flash. For Fusion, SmartFusion or SmartFusion2 production firmware (or at least a boot loader) will normally be stored in ENVM with the firmware image stored in an ENVM data storage client defined in the target hardware system design and programmed to the FPGA device using Libero/FlashPro.

[5] *E.g.* `production-..., boot-from-..., run-from-...`

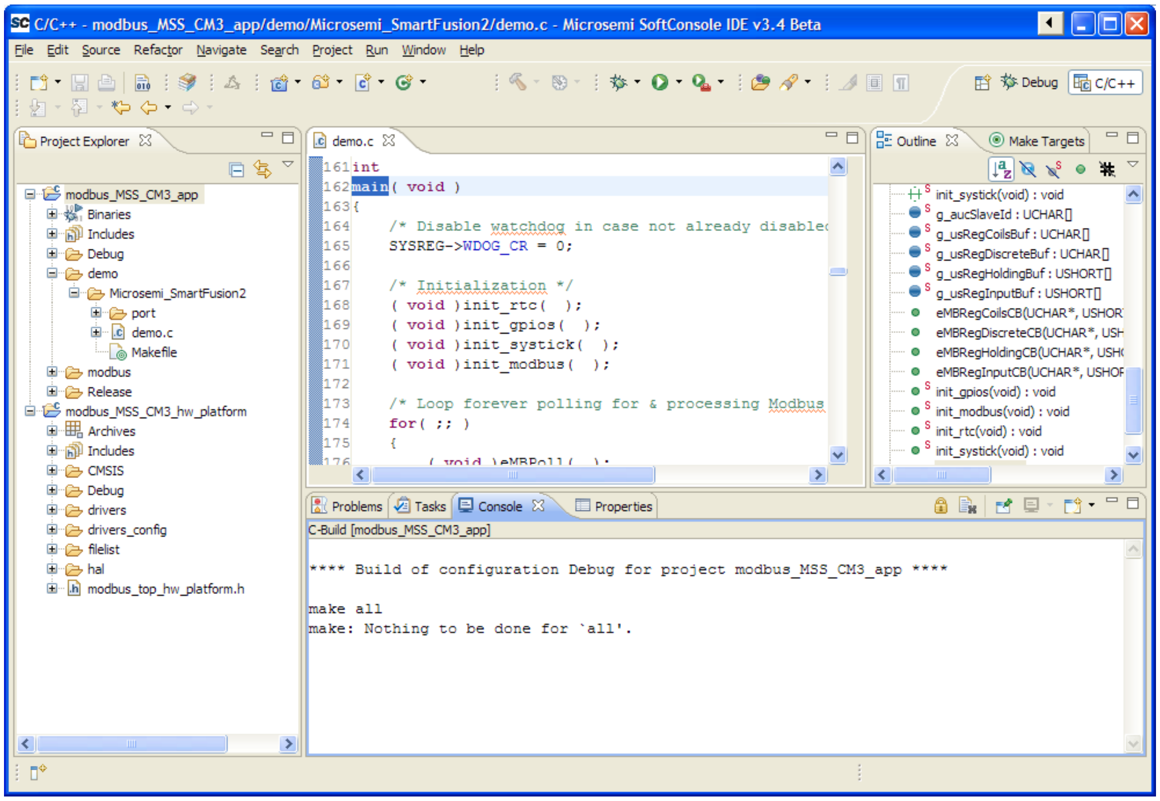
Eclipse concepts and GUI overview

Because the SoftConsole IDE is based on Eclipse and the Eclipse CDT familiarity with some key Eclipse concepts will lead to a better understanding of how SoftConsole works. Refer to the original Eclipse and CDT documentation for more detailed information about these and other related concepts:

- Eclipse Workbench User Guide: http://help.eclipse.org/galileo/index.jsp?nav=/0
- CDT C/C++ Development User Guide: http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.cdt.doc.user/concepts/cdt_o_home.htm

Workbench

The workbench is the main window that appears when SoftConsole is launched and a workspace has been opened. The workbench displays the main menu bar, toolbars, views and resources. The workbench can offer different perspectives as explained below. This is an example of the SoftConsole Eclipse workbench with the *C/C++* perspective active.

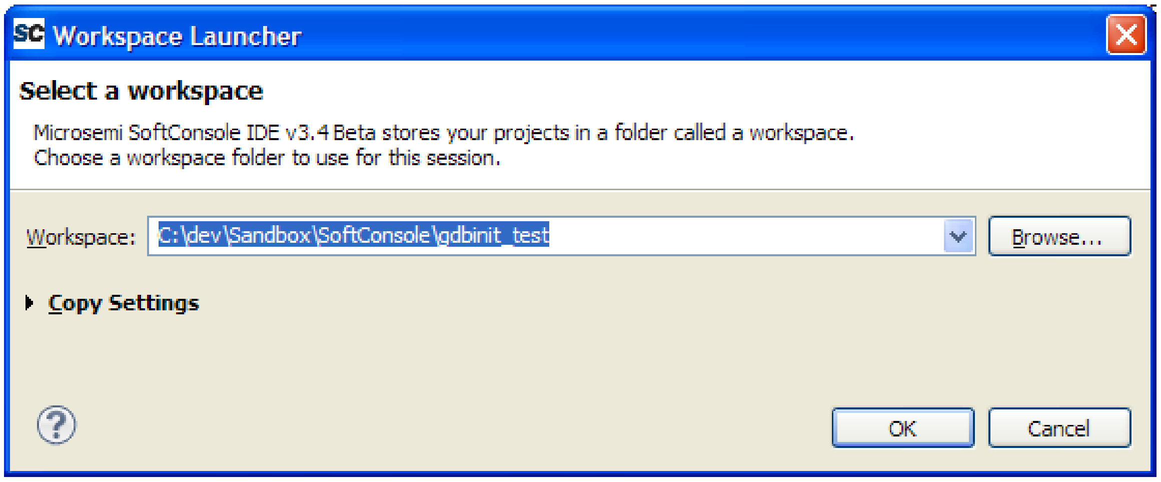

C/C++ - modbus_MSS_CM3_app/demo/Microsemi_SmartFusion2/demo.c - Microsemi SoftConsole IDE v3.4 Beta
File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
Debug
C/C++
Project Explorer
modbus_MSS_CM3_app
Binaries
Includes
Debug
demo
Microsemi_SmartFusion2
port
demo.c
Makefile
modbus
Release
modbus_MSS_CM3_hw_platform
Archives
Includes
CMSIS
Debug
drivers
drivers_config
filelist
hal
modbus_top_hw_platform.h
demo.c
Outline
Make Targets
init_systick(void) : void
g_aucSlaveId : UCHAR[]
g_usRegCoilsBuf : UCHAR[]
g_usRegDiscreteBuf : UCHAR[]
g_usRegHoldingBuf : USHORT[]
g_usRegInputBuf : USHORT[]
eMBRegCoilsCB(UCHAR*, USHOR
eMBRegDiscreteCB(UCHAR*, USH
eMBRegHoldingCB(UCHAR*, USH
eMBRegInputCB(UCHAR*, USHOR
init_gpios(void) : void
init_modbus(void) : void
init_rtc(void) : void
init_systick(void) : void
Problems
Tasks
Console
Properties
C-Build [modbus_MSS_CM3_app]

```
int
main( void )
{
    /* Disable watchdog in case not already disablec
    SYSREG->WDOG_CR = 0;

    /* Initialization */
    ( void )init_rtc(  );
    ( void )init_gpios(  );
    ( void )init_systick(  );
    ( void )init_modbus(  );

    /* Loop forever polling for & processing Modbus
    for( ;; )
    {
        ( void )eMBPoll(  );
```

```
**** Build of configuration Debug for project modbus_MSS_CM3_app ****

make all
make: Nothing to be done for `all'.
```

Workspace

The workspace is the main working folder which stores information about projects in the workspace, user preferences, cached data for Eclipse plug-ins etc. Multiple workspaces can be created but only a single workspace can be open at any one time. When using SoftConsole all operations take place in the context of the open workspace. A workspace can contain zero or more projects. When SoftConsole is launched it may prompt for a workspace to be selected or created.



Select a workspace

Microsemi SoftConsole IDE v3.4 Beta stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\dev\Sandbox\SoftConsole\gdbinit_test

The **File > Switch Workspace** menu option can be used to switch to a different workspace or to create a new one. To create a new workspace choose the **Other...** menu option and then create and select a new/empty folder on the file system.[6] The file system folder for a workspace contains a folder named `.metadata` which contains all of the SoftConsole/Eclipse data associated with the workspace.

Perspectives

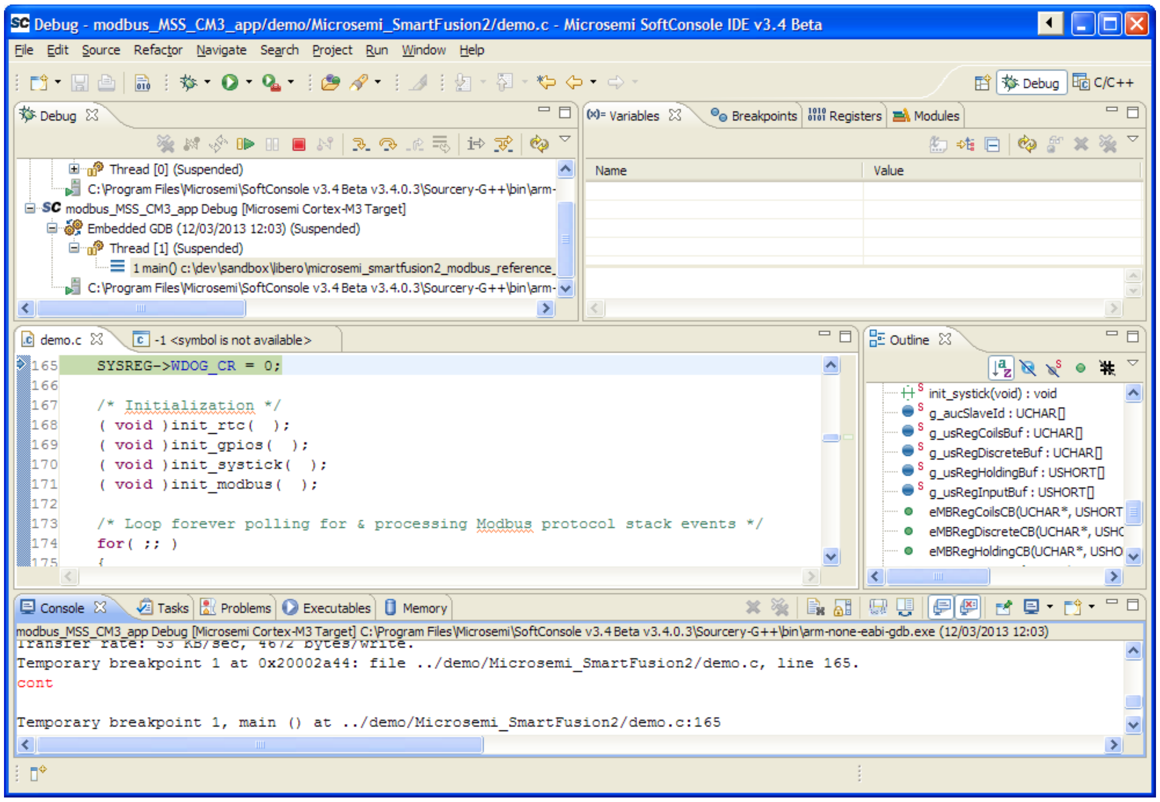
A perspective defines the initial set and layout of views in the workbench window. The workbench can offer multiple perspectives. Only one perspective can be active at any one time. The active perspective determines what views and resources are available and what operations can be carried out. For example the visible and enabled menu and toolbar options may differ depending on which perspective, view and/or resource is active/selected.

SoftConsole offers two perspectives relevant to the development of embedded firmware for Microsemi CPU based systems:

- The **C/C++** perspective – for managing and configuring projects, writing code, compiling projects etc.
- The **Debug** perspective – for interactive debugging of firmware on a hardware target.

The two perspectives display views and options appropriate and specific to these two modes of working. For example the default **C/C++** perspective is as pictured earlier and the default **Debug** perspective is as follows:

[6] Avoid using an existing non empty folder as a workspace. Avoid creating a workspace folder below an existing workspace or project folder.


Debug - modbus_MSS_CM3_app/demo/Microsemi_SmartFusion2/demo.c - Microsemi SoftConsole IDE v3.4 Beta
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
C/C++
Debug
(x)= Variables
Breakpoints
Registers
Modules
Name
Value
demo.c
-1 <symbol is not available>
Outline
Console
Tasks
Problems
Executables
Memory

```
Thread [0] (Suspended)
C:\Program Files\Microsemi\SoftConsole v3.4 Beta v3.4.0.3\Sourcery-G++\bin\arm-
modbus_MSS_CM3_app Debug [Microsemi Cortex-M3 Target]
  Embedded GDB (12/03/2013 12:03) (Suspended)
    Thread [1] (Suspended)
      1 main() c:\dev\sandbox\libero\microsemi_smartfusion2_modbus_reference_
  C:\Program Files\Microsemi\SoftConsole v3.4 Beta v3.4.0.3\Sourcery-G++\bin\arm-
```

```
    SYSREG->WDOG_CR = 0;

    /* Initialization */
    ( void )init_rtc(  );
    ( void )init_gpios(  );
    ( void )init_systick(  );
    ( void )init_modbus(  );

    /* Loop forever polling for & processing Modbus protocol stack events */
    for( ;; )
    {
```

```
init_systick(void) : void
g_aucSlaveId : UCHAR[]
g_usRegCoilsBuf : UCHAR[]
g_usRegDiscreteBuf : UCHAR[]
g_usRegHoldingBuf : USHORT[]
g_usRegInputBuf : USHORT[]
eMBRegCoilsCB(UCHAR*, USHORT
eMBRegDiscreteCB(UCHAR*, USHC
eMBRegHoldingCB(UCHAR*, USHO
```

```
modbus_MSS_CM3_app Debug [Microsemi Cortex-M3 Target] C:\Program Files\Microsemi\SoftConsole v3.4 Beta v3.4.0.3\Sourcery-G++\bin\arm-none-eabi-gdb.exe (12/03/2013 12:03)
Transfer rate: 53 KB/sec, 4672 bytes/write.
Temporary breakpoint 1 at 0x20002a44: file ../demo/Microsemi_SmartFusion2/demo.c, line 165.
cont

Temporary breakpoint 1, main () at ../demo/Microsemi_SmartFusion2/demo.c:165
```

The default views presented by each perspective are:

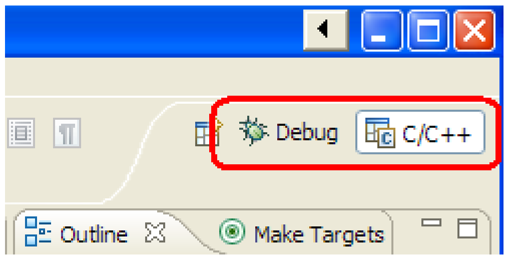
- **C/C++** perspective
  - On the left hand side of the workbench is the **Project Explorer** view which supports interaction with the projects in the current workspace.
  - In the middle of the workbench are zero or more editors allowing source and other files in any of the open projects to be viewed and edited. Multiple editors are displayed as a notebook (see below) of stacked tabs.
  - At the top right hand side of the workbench is a notebook containing
    - The **Outline** view which lists symbols in the currently active editor. Clicking on any symbol ion this view jumps to that symbol in the editor.
    - The **Make** view which lists make targets in the current workspace.
  - At the bottom right of the workbench is a notebook containing
    - The **Problems** view which displays build warnings and errors and supports quick navigation to the source of such problems.
    - The **Tasks** view
    - The **Console** view which displays project build and debug launch logging, status and error messages.
    - The **Properties** view

- **Debug** perspective
  - At the top left of the workbench is the **Debug** view displaying information, including the call stack, about debug sessions and tool buttons for various interactive debug operations (e.g. **Resume**, **Suspend**, **Terminate**, **Step Into**, **Step Over**, **Step Return**, **Instruction Stepping Mode** etc.)
  - At the top right of the workbench is a notebook containing

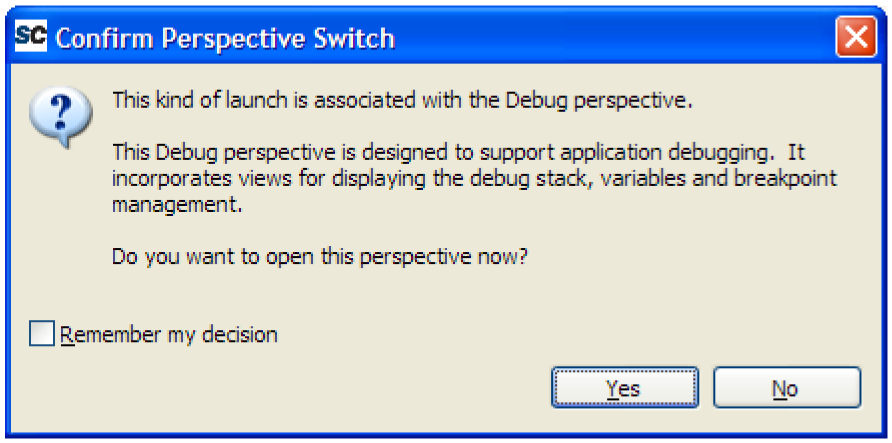
- The **Variables** view allowing local and other variables to be viewed and modified while debugging.
- The **Breakpoints** view allowing breakpoints to be viewed and modified. Note also that breakpoints can be managed via the editor window for a particular source file.
- The **Registers** view allowing target CPU/system registers to be viewed and modified.
- The **Modules** view

In addition to the default views presented by these two perspectives additional views can be used via the **Window > Show View** menu option. Additional views and other display settings are retained when changes are made. To revert a perspective to its original default appearance use **Window > Reset Perspective...**

The workbench presents buttons for selecting which perspective to use. This screenshot shows the buttons for the **C/C++** and **Debug** perspectives and indicates that the **C/C++** perspective is active:



SoftConsole can also automatically switch from the **C/C++** perspective to the **Debug** perspective when a debug session is launched. When this happens SoftConsole may prompt as follows:


Confirm Perspective Switch

This kind of launch is associated with the Debug perspective.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

☐ Remember my decision

When debugging is complete it is usually desirable to manually switch back to the **C/C++** perspective.

Views

Views are visual components within the workbench and are used to interact with, navigate and change the properties of resources. For example:

- The **Project Explorer** view in the **C/C++** perspective displays the list and hierarchy of projects in the current workspace and allows properties of each resource, folder or file to be viewed and modified.
- The **Debug** view in the **Debug** perspective displays details about debug sessions including the call stack.

Changes made in a view are saved immediately.

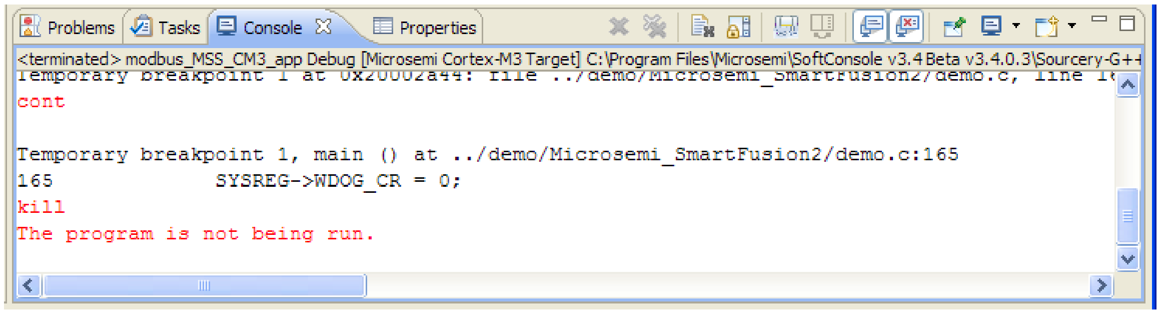
Editors

Editors are also visual components within the workbench and are used to edit source code and other files in projects. Changes made in an editor are only saved when explicitly requested.

Note that only a single view or can be active at any one time and the title bar or tab of the active view or editor is highlighted.

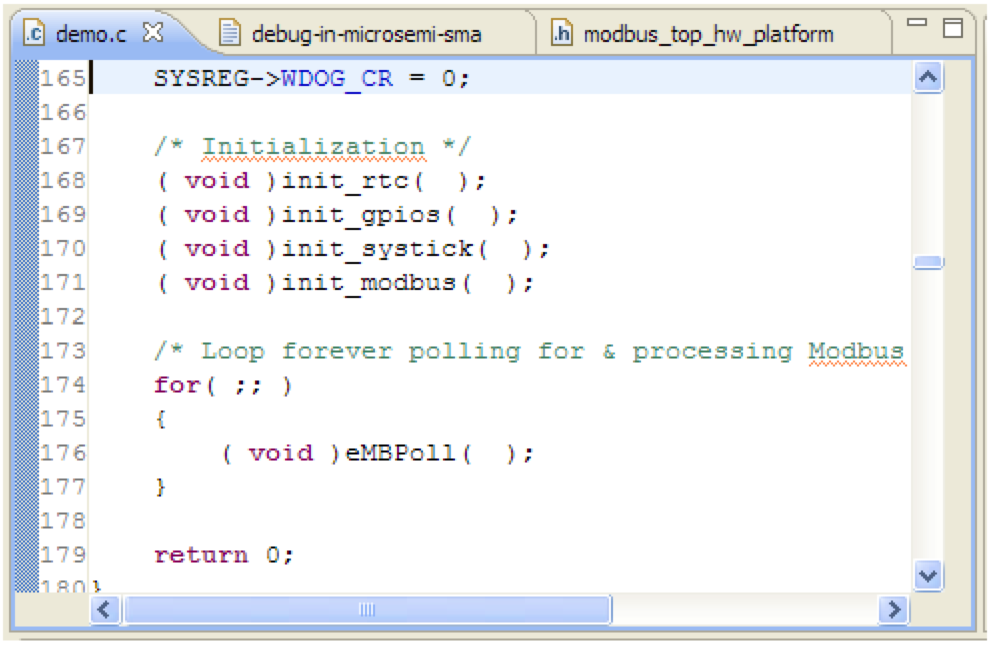
Notebooks

A notebook is simply a stacked arrangement of tabbed views. For example this is a notebook showing the **Problems**, **Tasks**, **Console** and **Properties** views with the **Console** view active:



```
<terminated> modbus_MSS_CM3_app Debug [Microsemi Cortex-M3 Target] C:\Program Files\Microsemi\SoftConsole v3.4 Beta v3.4.0.3\Sourcery-G++
Temporary breakpoint 1 at 0x20002a44: file ../demo/Microsemi_SmartFusion2/demo.c, line 16
cont

Temporary breakpoint 1, main () at ../demo/Microsemi_SmartFusion2/demo.c:165
165             SYSREG->WDOG_CR = 0;
kill
The program is not being run.
```

And this is a notebook displaying three editors with the `demo.c` editor active:



```
    SYSREG->WDOG_CR = 0;

    /* Initialization */
    ( void )init_rtc(  );
    ( void )init_gpios(  );
    ( void )init_systick(  );
    ( void )init_modbus(  );

    /* Loop forever polling for & processing Modbus
    for( ;; )
    {
        ( void )eMBPoll(  );
    }

    return 0;
}
```

Resources

Resources are the projects, folders and files that existing in the workbench.

Files

The files contained in a project – e.g. C/C++ and assembler source code and include files, linker scripts, compiled object files, linked executable files, generated list and map and hex files etc.

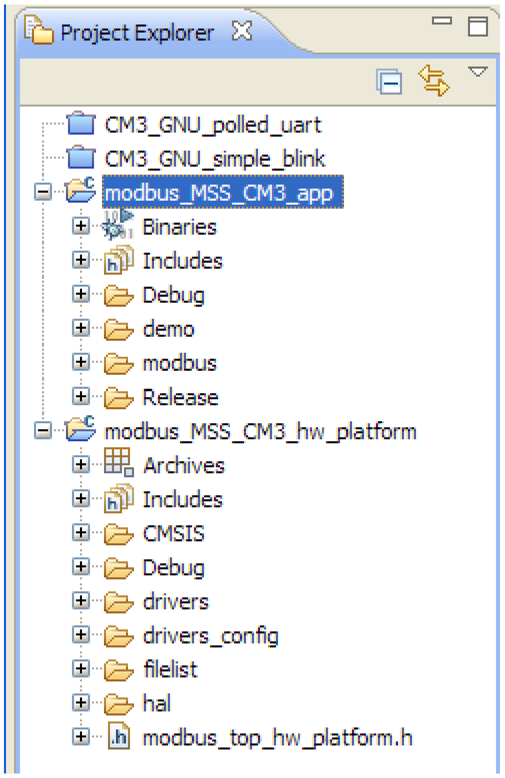
Folders

In the workbench folders are contained inside projects or inside other folders and folders can contain other folders and/or files.

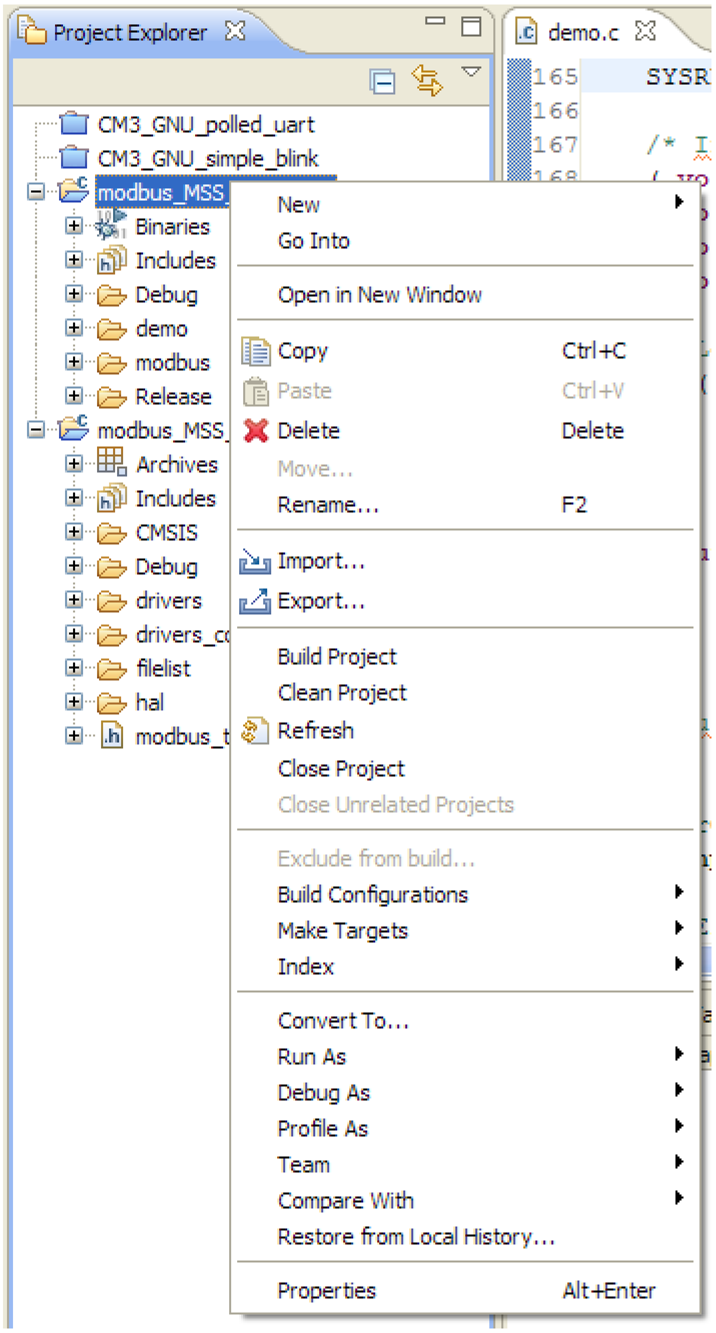
Projects

A project contains folders and files and is used for building, version management, sharing and resource organization. Projects map to folders on the underlying file system. The file system folder containing a SoftConsole project contains two files named `.project` and `.cproject` along with the other folders and files that appear in the project in SoftConsole.

The **Project Explorer** view displays the list of projects in the current workspace. For example this screenshot shows the **Project Explorer** listing four projects in the current workspace:



Note that the first two projects are closed while the other two projects are the application and library projects generated by Libero from the associated hardware project. Many commonly used project operations can be carried out by left mouse button clicking on a project in the **Project Explorer** to select it and then choosing the appropriate menu option or toolbar button, or by right mouse button clicking on the project and choosing an option from the context menu that appears:


Project Explorer
demo.c
CM3_GNU_polled_uart
CM3_GNU_simple_blink
modbus_MSS
Binaries
Includes
Debug
demo
modbus
Release
modbus_MSS_
Archives
Includes
CMSIS
Debug
drivers
drivers_c
filelist
hal
modbus_t
New
Go Into
Open in New Window
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Move...
Rename... F2
Import...
Export...
Build Project
Clean Project
Refresh
Close Project
Close Unrelated Projects
Exclude from build...
Build Configurations
Make Targets
Index
Convert To...
Run As
Debug As
Profile As
Team
Compare With
Restore from Local History...
Properties Alt+Enter

Using SoftConsole

Not all of the capabilities of SoftConsole are covered in detail here as many of these are already covered adequately or in more detail elsewhere – e.g. in the Eclipse and CDT or other component documentation cited earlier.

Bear in mind that often there is more than one way to perform a particular operation – e.g. using the main menu, a toolbar button, a context specific menu etc.

Getting help

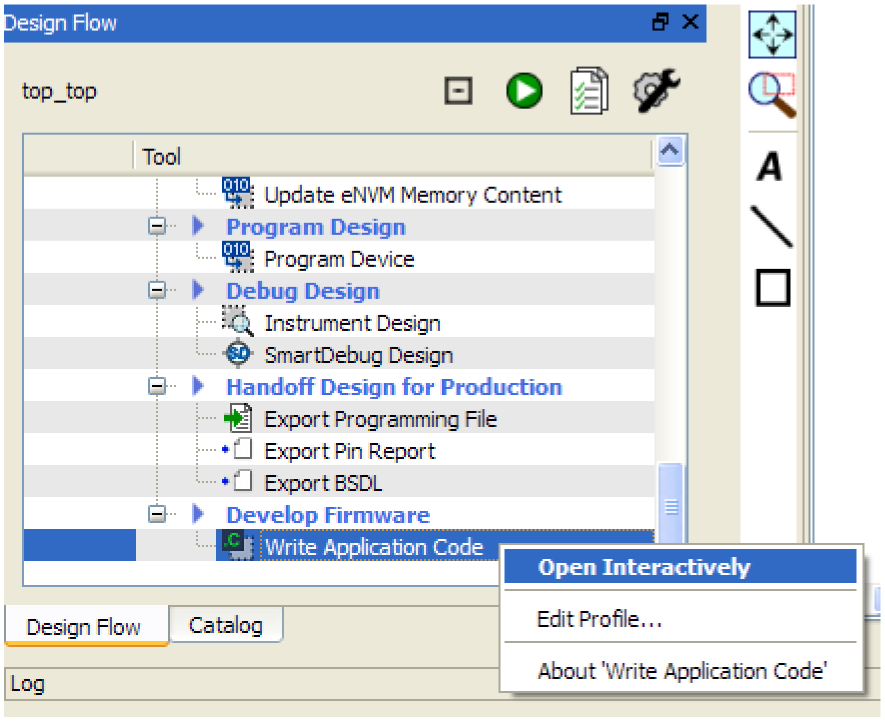
General help about Eclipse and CDT is available from the **Help** option in the main menu bar. This is useful for general guidance on Eclipse and CDT concepts and capabilities. The documentation mentioned earlier for the various components bundled with SoftConsole should also be referenced for details of how these tools work, what options are available and so on.

Launching SoftConsole

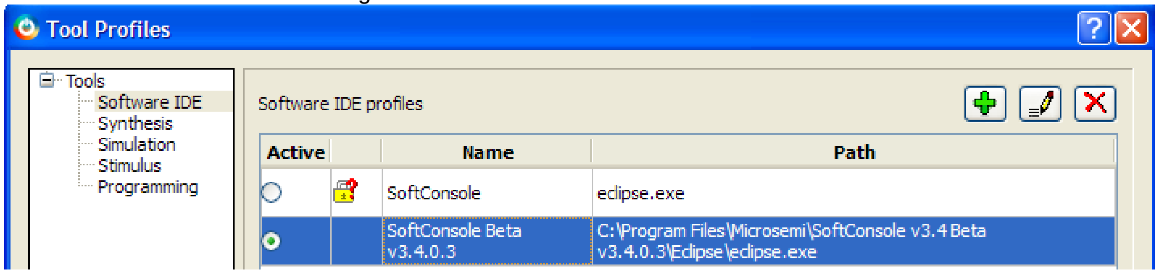
SoftConsole can be launched in a number of ways:

- From the shortcut that the installer creates on the Windows desktop or from the Windows **Start** menu (e.g. **Start > All Programs > Microsemi SoftConsole > Microsemi SoftConsole IDE**).
- By executing `C:\Program Files\Microsemi\SoftConsole ...\Eclipse\eclipse.exe`[7].
- From Libero SoC using **Design Flow > Write Application Code > Open Interactively**

[7] *`C:\Program Files\Microsemi\SoftConsole ...` is the default installation folder so adjust this if SoftConsole is installed in another location or was installed alongside Libero by the Libero installer.*



when SoftConsole has been configured as the **Software IDE Tool Profile**

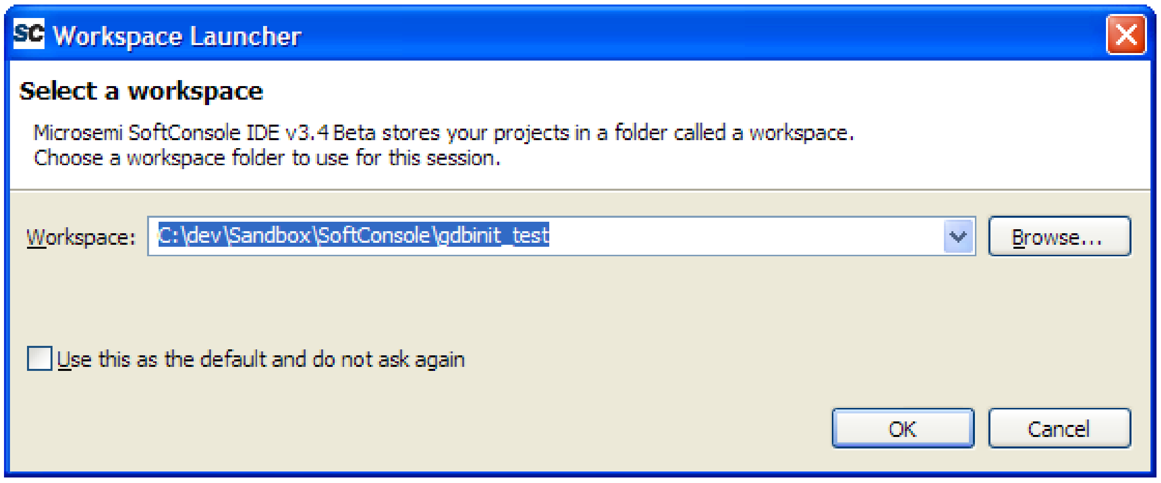


Active		Name	Path
○		SoftConsole	eclipse.exe
●		SoftConsole Beta v3.4.0.3	C:\Program Files\Microsemi\SoftConsole v3.4 Beta v3.4.0.3\Eclipse\eclipse.exe

and Libero has generated the SoftConsole workspace and application/library projects matching the hardware design being created.

Managing workspaces

When SoftConsole is launched it may prompt for a workspace:


Workspace Launcher

Select a workspace

Microsemi SoftConsole IDE v3.4 Beta stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\dev\Sandbox\SoftConsole\gdbinit_test

From the **Workspace Launcher** dialog an existing workspace folder (containing a `.metadata` folder) can be selected or a new/blank folder can be selected in order to create a new/blank workspace.

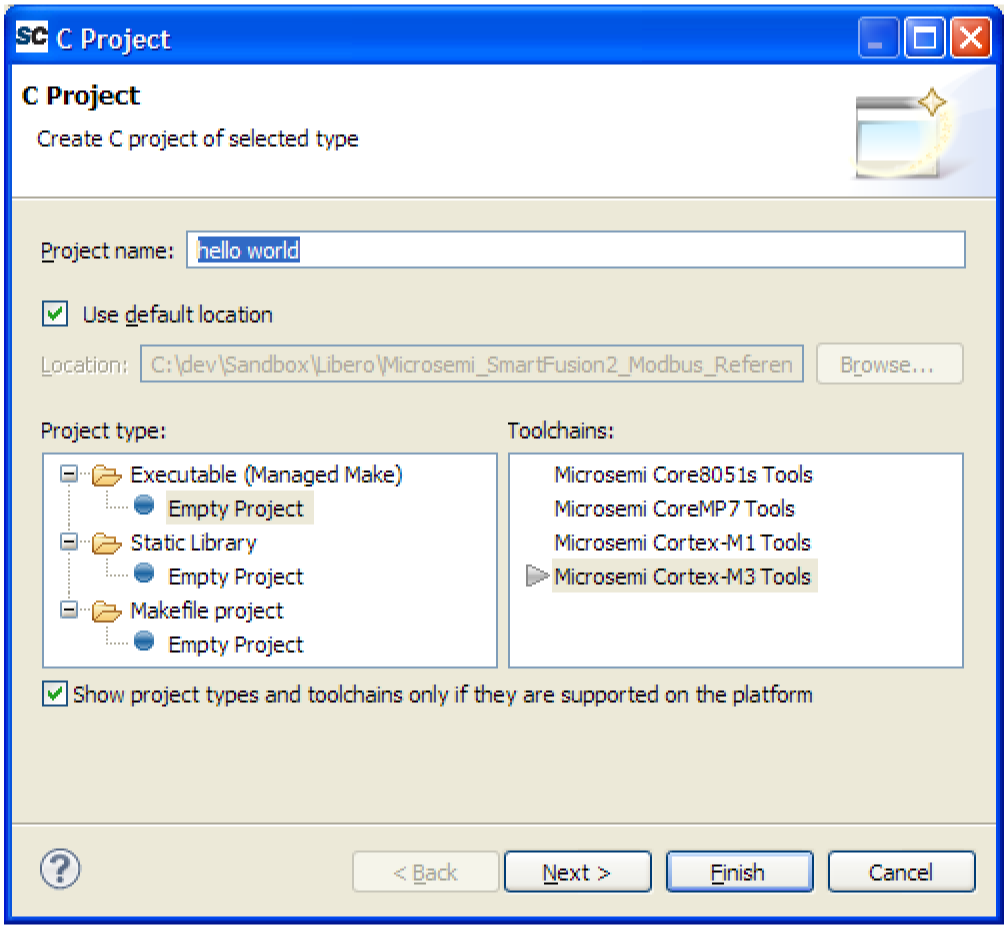
SoftConsole will not prompt for a workspace if one was previously selected and is still accessible and the **Use this as the default and do not ask again** option was checked or if it is launched from the Libero SoC **Design Flow**.

To switch to or create another workspace when one is already open use the **File > Switch Workspace** menu option.

Managing projects

Creating projects

A new/blank project can be created using **File > New > C Project** or **File > New > C++ Project** which presents the **C/C++ Project** dialog:


C Project

C Project

Create C project of selected type

Project name: hello world

✓ Use default location

Location: C:\dev\Sandbox\Libero\Microsemi_SmartFusion2_Modbus_Referen

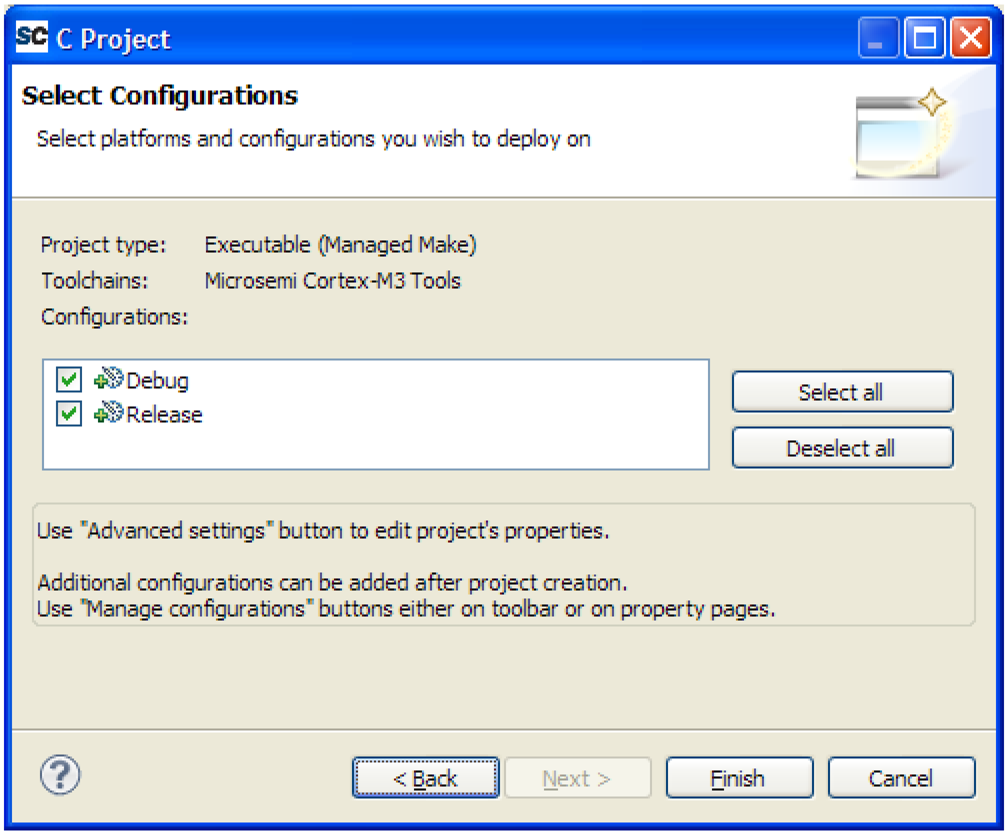
Project type:
- Executable (Managed Make)
  - Empty Project
- Static Library
  - Empty Project
- Makefile project
  - Empty Project

Toolchains:
- Microsemi Core8051s Tools
- Microsemi CoreMP7 Tools
- Microsemi Cortex-M1 Tools
- Microsemi Cortex-M3 Tools

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

Enter/select the appropriate settings for the new project:

- **Project name**: the name for this project
- **Use default location**: leave this checked in order to create the project in the current workspace
- **Project type**: select the appropriate project type
  - **Executable (Managed Make) > Empty Project**: a project which builds an executable program image and whose make/build process is completely managed by SoftConsole.
  - **Static Library > Empty Project**: a project which builds a library which can be linked to an application project and whose make/build process is completely managed by SoftConsole.
  - **Makefile project > Empty Project**: a project whose build process is not managed by SoftConsole but by a makefile.
- **Toolchains**: the target CPU toolchain.
- Clicking **Next>** presents the **Select Configurations** dialog.


C Project

Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Executable (Managed Make)

Toolchains: Microsemi Cortex-M3 Tools

Configurations:

- [✓] Debug
- [✓] Release

Use "Advanced settings" button to edit project's properties.

Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.

By convention SoftConsole projects contain two build configurations – Debug and Release – and it is normally advisable to accept and use these.

A new/blank project created in this way is ready to be populated with the necessary source files and folders and have its project properties configure appropriately. As mentioned earlier selecting the project in the Project Explorer view, launching the Firmware Catalog from SoftConsole and generating individual firmware cores generates the driver files into the project. Depending on the firmware core some include directory settings may need to be configured in the project properties.

An often more convenient alternative to starting with a new/blank project is to take an existing project and adapt it to the needs of a specific target system. This can be done by allowing Libero SoC to generate the SoftConsole workspace and application/library projects matching the target hardware system as mentioned earlier, or by generating a suitable firmware core example project and importing it into the SoftConsole workspace.

Importing and exporting projects

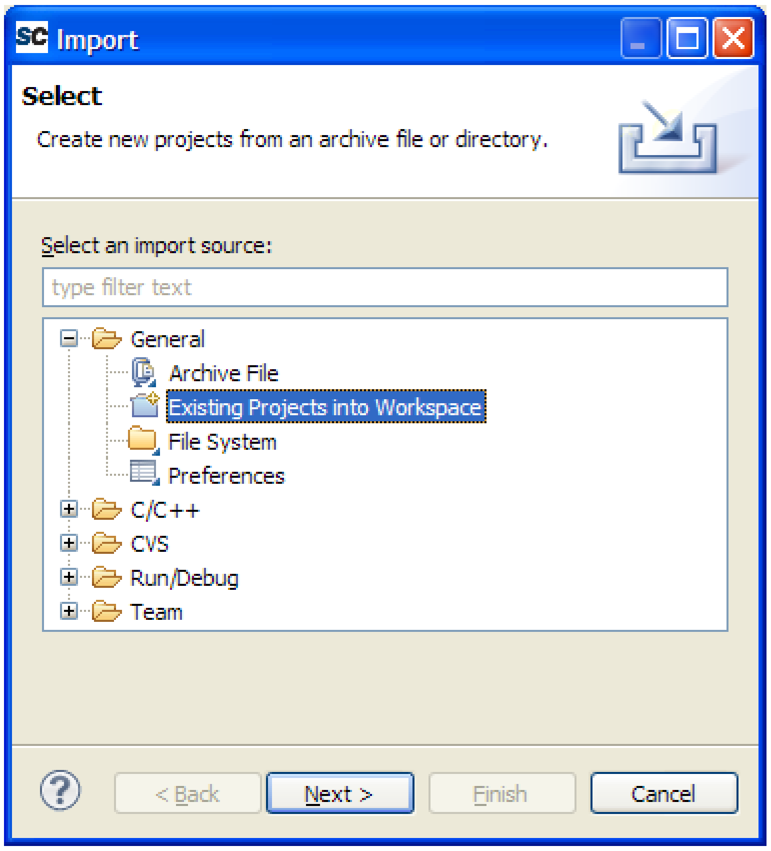
As mentioned earlier the Firmware Catalog can be launched from SoftConsole and firmware cores generated into the selected project. Similarly the Firmware Catalog can generate fully self contained and working example projects[8] from firmware cores into the current workspace. When this is done, although the generated project is in the workspace folder the project still needs to be imported into the workspace.

---

[8] *Refer to the firmware core documentation and example project header files for information about any hardware requirements assumed by the example project.*

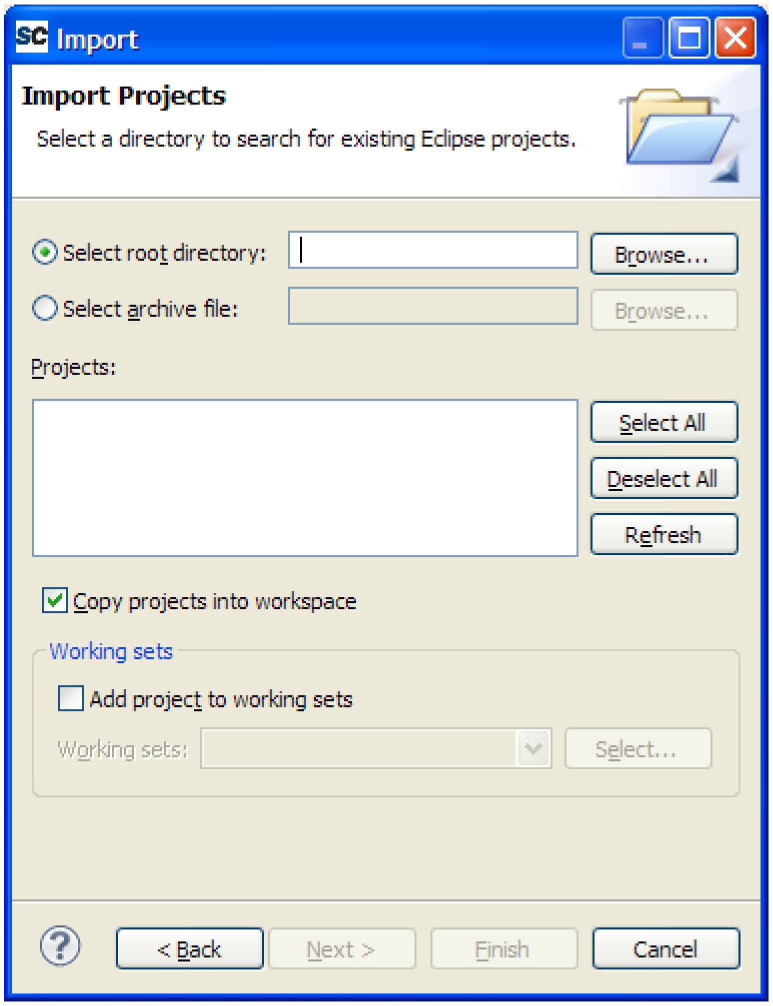
To import a project:

- Choose the **File > Import...** menu option or
- Right click inside the **Project Explorer** view and from the context menu choose **Import...**
- The **Import > Select** dialog appears

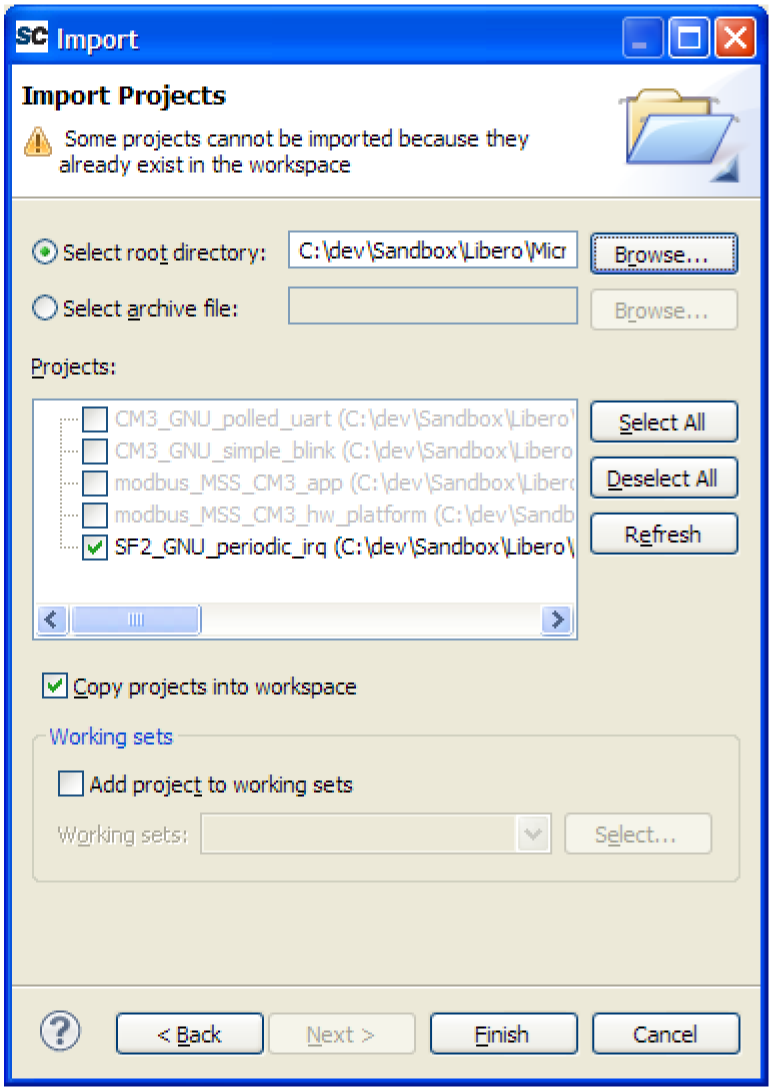


- Select **General > Existing Projects into Workspace** and click **Next>**.

- The **Import > Import Projects** dialog appears:



- Select **Select root directory** and click **Browse...** which presents the **Browse For folder** dialog which is already located at the current workspace's file system folder.
- If the project is to be imported from some other location then browse to the required folder.
- Alternatively if the project is to be imported from an archive file select **Select archive file** and then click its **Browse...** button to locate the required archive file.
- When the source folder or archive file for the project(s) to be imported has been selected then the **Projects** list in the **Import** dialog lists the available project(s). Projects that are already in the workspace are greyed out.



Import

Import Projects

Some projects cannot be imported because they already exist in the workspace

Select root directory: C:\dev\Sandbox\Libero\Micr

Select archive file:

Projects:

- [ ] CM3_GNU_polled_uart (C:\dev\Sandbox\Libero\
- [ ] CM3_GNU_simple_blink (C:\dev\Sandbox\Libero
- [ ] modbus_MSS_CM3_app (C:\dev\Sandbox\Liberc
- [ ] modbus_MSS_CM3_hw_platform (C:\dev\Sandb
- [✓] SF2_GNU_periodic_irq (C:\dev\Sandbox\Libero\

[✓] Copy projects into workspace

Working sets

[ ] Add project to working sets

Working sets:

- Select the required project(s) and check **Copy projects into workspace**.
- Click **Finish** and the required project(s) are imported into the workspace and appear in the **Project Explorer** view.

Once imported a firmware core example project can be used as-is or as the basis for another program by adapting the project contents and properties as required.

To export a project:

- Select the **File > Export...** menu option or
- Right click in the **Project Explorer** view or on a project and from the context menu choose **Export...**
- In the **Export > Select** dialog select **General > Archive File** to export to an archive file or **General > File System** to export to the file system.
- Click **Next>**
- Select the projects , folders, files etc. to export and the other export options as required.
- Click **Finish** to export.

Opening, closing and deleting projects

Sometimes when there are several projects in a workspace it may be convenient to close those that are not in use at a specific point in time. To do this select a project or multi-select multiple projects, right click in the **Project Explorer** view and from the context menu choose **Close Project**.

Alternatively select the project to be kept open (along with any related projects), right click in the **Project Explorer** view and from the context menu choose **Close Unrelated Projects**.

To delete a project select it, right click in the **Project Explorer** view and from the context menu choose **Delete**. The **Delete Resources** dialog appears. Check the **Delete project contents on disk (cannot be undone)** to delete the project from the SoftConsole workspace and also delete all underlying project files on the file system. Click **OK** to carry out the deletion operation.

Other common operations on projects

Select a project and then right click in the **Project Explorer** view to see the context menu which contains other operations that can be carried out on projects. For example:

- **Copy** and **Paste** projects in the workspace.
- **Rename...** a project in the workspace – note that this just changes the workspace name of the project but does not change the name of any build targets etc.
- **Refresh** the project which may be needed, for example, if files are copied to a project via the file system and they do not automatically show up in the SoftConsole **Project Explorer** view.

Build related project operations are dealt with later.

Project build settings

The project build settings specify details of the number and type of build configurations supported by a project and the tools and configuration options used to build these. These will vary from project to project but the Firmware core example projects and Libero generated application/library projects provide useful examples of how the project build settings can be configured to create debug and release executable images and are a useful place to start when creating firmware for a CPU based system.

The project build settings can be accessed as follows:

- Select the project in the **Project Explorer** view.
- Type **Alt + Enter** or choose **File > Properties** or right click and from the context menu select **Properties**.
- The **Properties for <project-name>** dialog appears.
- Browse to **C/C++ Build > Settings > Tool Settings**.
- The **Configuration:** setting lists the build configurations supported by this project. Each build configuration has its own set of tool settings although it is possible to share settings between all configurations where necessary by choosing **[All configurations]** from the dropdown list.
- The **Tool Settings** property page lists all of the tools used to build a program such as the compiler, assembler, linker etc.
- Different tools have different categories of settings and these can be tuned as required to suit the needs of a specific firmware project and build configuration.

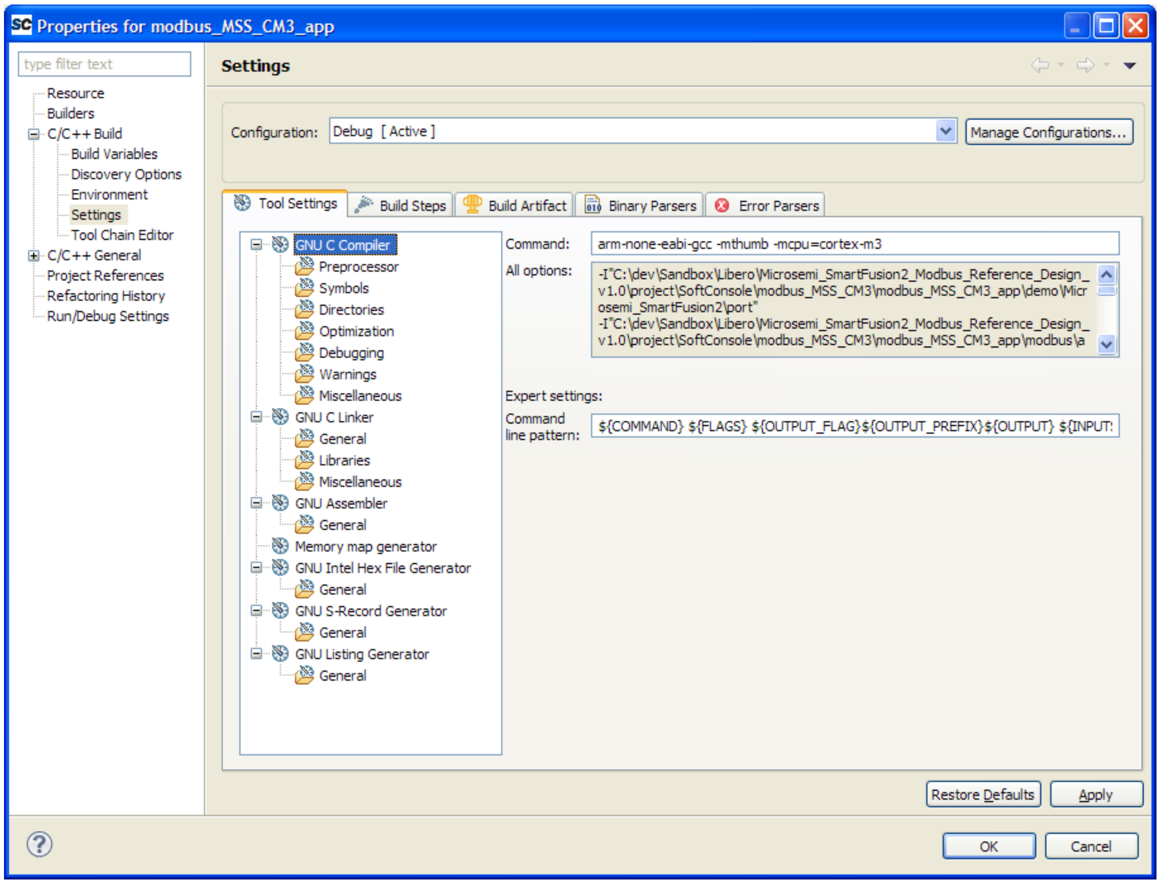
The project build settings user interface and the tools listed differ depending on a number of factors:

- The target CPU type (ARM or 8051).
- The build configuration (e.g. debug or release)
- The project type (e.g. C or C++, application or library)

Note: please refer to the relevant tool specific documentation for details of the configuration options supported by each tool – e.g. GCC, GNU Linker, GNU Assembler, SDCC tools etc.

ARM projects

This is an example of the project build settings user interface for an ARM project.



Configuration: Debug [Active]

Command: arm-none-eabi-gcc -mthumb -mcpu=cortex-m3

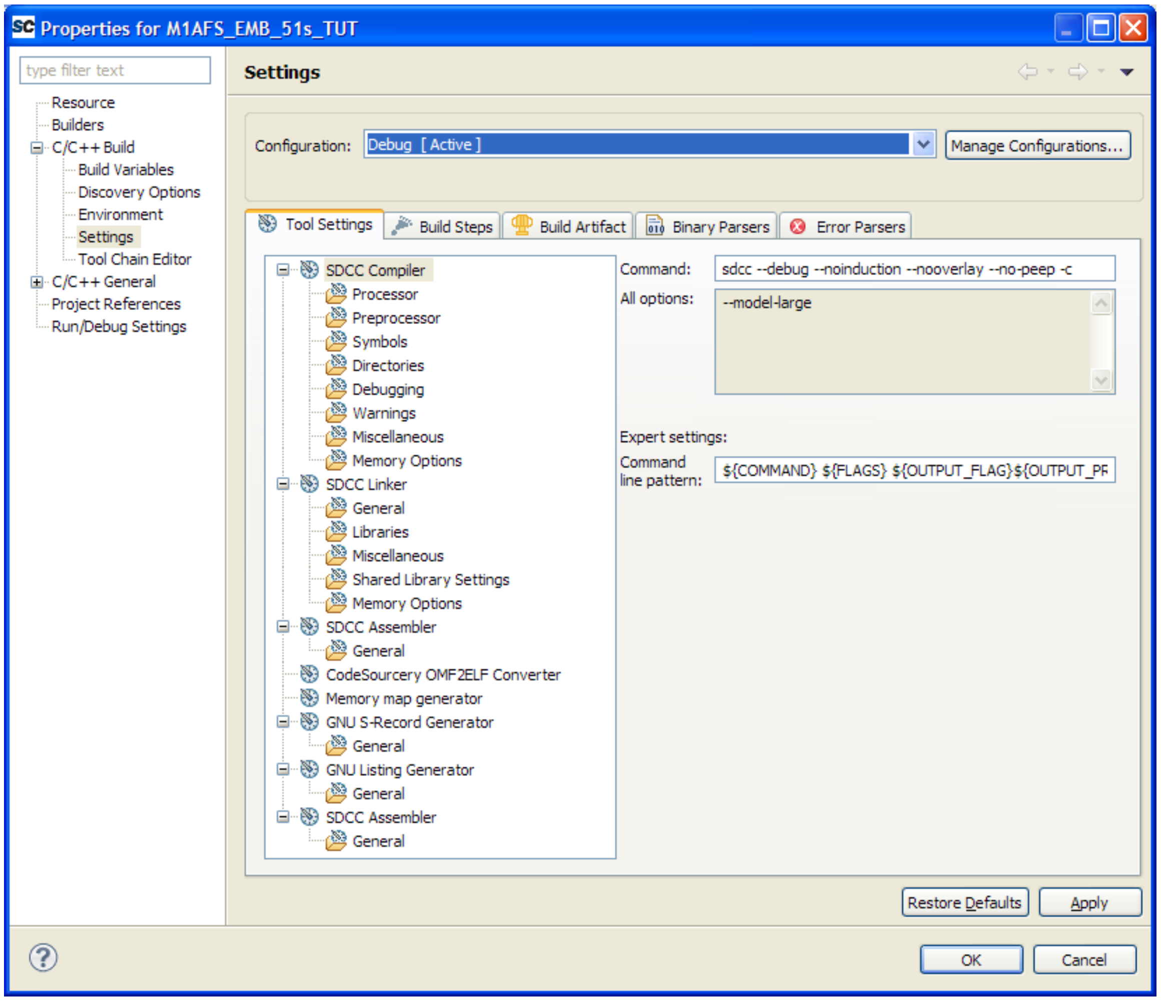
All options:
```
-I"C:\dev\Sandbox\Libero\Microsemi_SmartFusion2_Modbus_Reference_Design_
v1.0\project\SoftConsole\modbus_MSS_CM3\modbus_MSS_CM3_app\demo\Micr
osemi_SmartFusion2\port"
-I"C:\dev\Sandbox\Libero\Microsemi_SmartFusion2_Modbus_Reference_Design_
v1.0\project\SoftConsole\modbus_MSS_CM3\modbus_MSS_CM3_app\modbus\a
```

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX}${OUTPUT} ${INPUT:

8051 projects

This is an example of the project build settings user interface for an 8051 project:



Configuration: Debug [Active]

Command: sdcc --debug --noinduction --nooverlay --no-peep -c

All options: --model-large

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PR

Application and library project dependencies

The workspace generated by Libero for a CPU based system contains an application and a library project. The library project contains all firmware drivers and target system specific settings. The application project is intended to be filled out by the user with the custom firmware required by the system. The application project is configured to have a dependency on the library project so that when the application is built then the library is also rebuilt if needed so that it is available to be linked into the application. The same build configuration (e.g. debug or release) should be selected for both projects to ensure that this works smoothly. The application project build settings contain the necessary include directory and library paths to ensure that the library project links correctly. The application project properties also specify the dependency on the library project under **Properties > Project References** where the library project in the workspace is selected.

This approach to having application projects that depend on one or more library projects in the same workspace can be adopted for use with other projects.

Building a project

Build configurations

By convention SoftConsole projects normally contain two build configurations with the following characteristics:

- Debug
  - Facilitates interactive and iterative download to and debugging on the target hardware.
  - Most or all optimization disabled (e.g. GCC `-O0` option or equivalent).
  - Most or all debugging symbolic information enabled (e.g. GCC `-g3` option or equivalent).
  - Usually linked with linker settings or a linker script that facilitates download to and debugging from embedded or external RAM
  - Optionally linked to facilitate download to and debugging from embedded or external flash memory.

- Release
  - Facilitates the creation of production program images for storage in and execution from a suitable internal or external flash memory.
  - Most or all optimization enabled (e.g. GCC optimization options `-O2`, `-O3` or `-Os` or equivalent).
  - Debugging disabled (e.g. no GCC `-g` option or equivalent).
  - Usually linked with one of the CMSIS/HAL "production" linker scripts or similar.[9]
  - Optionally linked with a linker script that facilitates interactive download to and debugging on the target platform but the presence of optimization and/or the lack of debug symbolic information may militate against accurate debugging (e.g. line tracking).

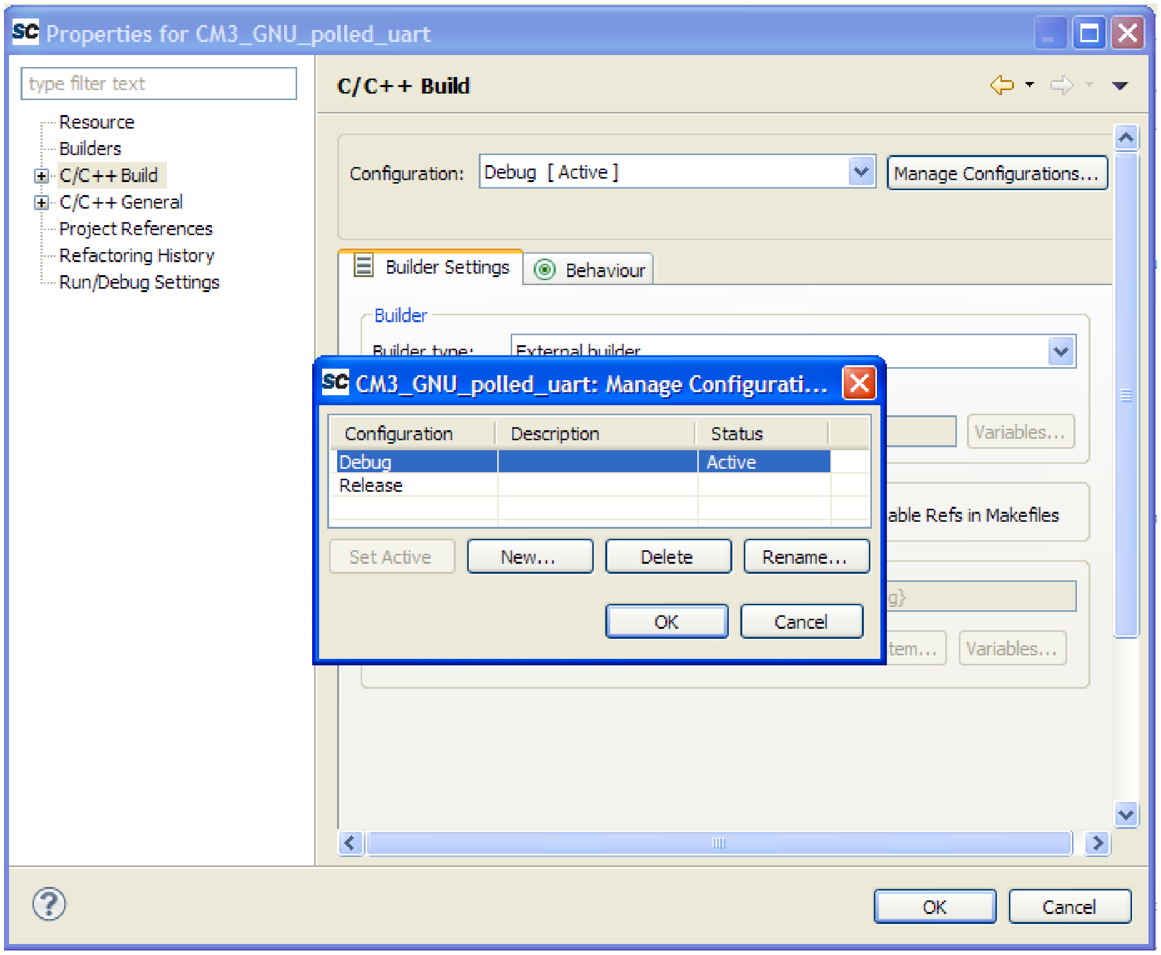
However bear in mind that these build configurations are simply conventional and it is perfectly feasible to have as many and as varied build configurations as required by a specific program and target system. For example having different debug or release configurations with different `#defines` to enable or disable optional functionality.

To add, remove or modify build configurations supported by a project:

- Select the project in the **Project Explorer** view and choose the **File > Properties** menu option or
- Right click on the project in the **Project Explorer** view and select **Properties** from the context menu
- Select **C/C++ Build**

[9] *E.g.* `production-..., boot-from-..., run-from-...`

- Click **Manage Configurations...**


Properties for CM3_GNU_polled_uart

C/C++ Build

Configuration: Debug [Active]

CM3_GNU_polled_uart: Manage Configurati...

Configuration	Description	Status
Debug		Active
Release		

Matching firmware to target hardware

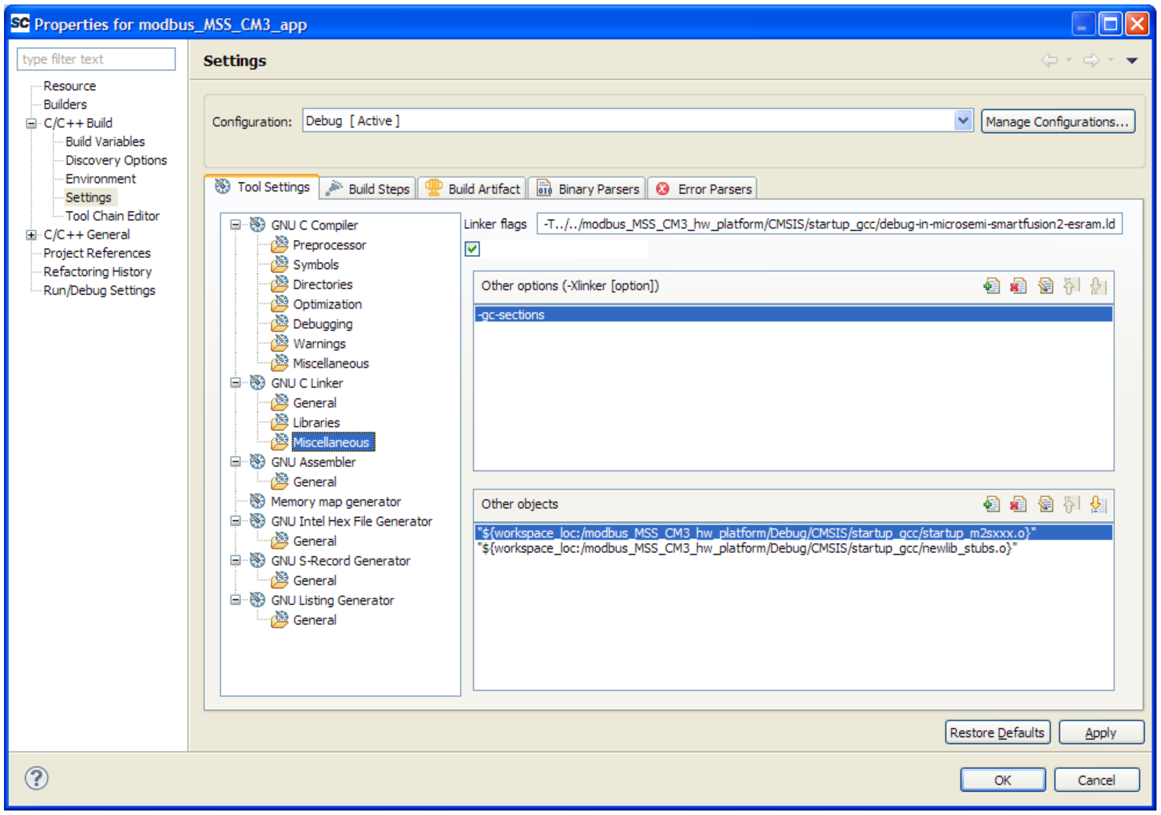
It is obviously important to match the firmware to the target hardware. Libero can assist in this respect by generating the appropriate workspace and application/library projects, project settings and firmware drivers which are then picked up when SoftConsole is launched from Libero.

For SmartFusion and SmartFusion2 Cortex-M3 based systems it is essential that the `<libero-project-root>/firmware/drivers_config` folder and contents are copied into the firmware project if the Libero generated workspace and application/library projects are not being used.

Memory map and linker scripts

And essential part of matching the firmware to the target hardware is ensuring that the firmware knowledge of the memory map matches the actual target hardware. This includes the location and size of each memory region and the address of each memory mapped peripheral.

For ARM CPU based systems it is important to specify a suitably linker script in the project build settings:



Configuration: Debug [Active]

Linker flags: -T../../modbus_MSS_CM3_hw_platform/CMSIS/startup_gcc/debug-in-microsemi-smartfusion2-esram.ld

Other options (-Xlinker [option])

```
-gc-sections
```

Other objects

```
"${workspace_loc:/modbus_MSS_CM3_hw_platform/Debug/CMSIS/startup_gcc/startup_m2sxxx.o}"
"${workspace_loc:/modbus_MSS_CM3_hw_platform/Debug/CMSIS/startup_gcc/newlib_stubs.o}"
```

Very often the CMSIS/HAL example linker scripts can be used either as-is or having been adapted to match the needs of a specific target system.

For 8051 CPU based systems the SDCC toolchain does not use linker scripts but uses tool configuration options to specify details of the memory model and memory map to be used – for example under **Properties > C/C++ Build > Settings > Tool Settings > SDCC Compiler/SDCC Linker > Memory Options**.

Building projects

There are a number of ways to build projects in SoftConsole. One simple way is to specify the active build configuration as follows:

- Select the project in the **Project Explorer** view.
- From the main menu bar choose **Project > Build Configurations > Set Active** and choose one of the listed build configurations.[10]

Thereafter many of the build related actions apply to the active build configuration. For example to build the active build configuration:

[10] When an application project depends on a library project – such as with the Libero generated workspace and application/library projects – the same build configuration must be selected for both the application and library projects in order for the application to build and link correctly.

- Select the project in the **Project Explorer** view.
- Right click on the project and from the context menu choose **Build Project**.

To clean the project:

- Select the project in the **Project Explorer** view.
- Right click on the project and from the context menu choose **Clean Project**.

An alternative way to clean a project is:

- In the **Project Explorer** view select the build configuration folder in the project (e.g. **Debug** or **Release**).
- Right click on the folder and from the context menu select **Delete**.
- Click **OK** on the **Delete Resources** dialog box.

To build some or all defined build configurations for a project rather than just the active build configuration:

- Select the project in the **Project Explorer** view.
- Right click on the project and from the context menu choose **Build Configurations > Build > All** to build all defined build configurations or **Build Configurations > Build > Select** to build selected build configurations.

To build the active build configurations for all open projects:

- From the main menu bar select **Project > Build All**.

A project can also be configured to build every time anything related to the project changes – e.g. project properties are changed or file changes are saved in an editor:

- Select the project in the **Project Explorer** view.
- From the main menu bar choose **Project > Build Automatically**.

Build artifacts

Build artifacts are files produced by the build process and stored in the build configuration folder in the project – e.g. Debug or Release folder for the respective build configurations. This section briefly summarises some of build artifacts for ARM and 8051 projects.

ARM and 8051 projects

- ELF (Executable and Linkable Format) executable file (no file extension or `*.elf`): The linked executable program image in ELF format which may include symbolic debug information. The debugger uses this file when loading and debugging a program. For ARM projects the ELF file is produced by the GNU ld linker. For 8051 projects the ELF file is produced from the AOMF51 file by the **CodeSourcery OMF2ELF Converter** build step.
- Library archive file (`*.a` or `*.lib`): The library archive file for a library project which can be linked into an application project. The Libero generated workspace for a CPU based system illustrates how application and library projects can be used.
- Object files (`*.o` or `*.rel`): Object files produced by the compiler which may be linked into an executable or archived into a library. There is normally one object file per compilation unit or source file. Object files may be stored in a folder hierarchy beneath the build configuration target folder (e.g. Debug or Release).
- Intel Hex file (`*.hex`): An Intel Hex format file (http://en.wikipedia.org/wiki/Intel_HEX) for the executable program image. This is generated from the ELF format program image by the **GNU Intel Hex File**

Generator build step which calls `arm-none-eabi-objcopy -Oihex <elf-file>`. This file can be programmed to non volatile memory using a suitable flash programmer including FlashPro to program an ENVM data storage client.

- Motorola S-record file (`*.srec`): A Motorola S-record format file (http://en.wikipedia.org/wiki/SREC_%28file_format%29) for the executable program image. This is generated from the ELF format file by the **GNU S-Record Generator** build step which calls `arm-none-eabi-objcopy -Osrec <elf-file>`. This file can be programmed to non volatile memory using a suitable flash programmer.
- Makefiles (`makefile, *.mk`): Make/build files created by a **Managed Make** project and used to build a project build configuration.
- `memory-map.xml`: The memory map file created by the **Memory map generator** build step which calls `actel-map`. This file is read by the debug sprite in order to obtain information about the target memory – in particular if flash programming needs to be used for any memory space.

ARM projects

- List file (`*.lst`): A list file for the program image. This is generated from the ELF format program image by the **GNU Listing Generator** build step which calls `arm-none-eabi-objdump -h -S <elf-file>`. The list file contains section information and a disassembly listing for the program image.
- Map file (`*.map`): A map file for the program image. This is generated by default when **Project Properties > C/C++ Build > Settings > Tool Settings > GNU C Linker > Miscellaneous > Generate linker memory map file** is checked. The map file contains size and location information for sections and headers in the program image.

  The list and map files are useful for analyzing the contents of an executable program image such as when investigating ways of reducing the program image size by eliminating unnecessary code.

For further details about GCC build artifacts, intermediate files/formats and commands and options refer to the GCC and related documentation.

8051 projects

- Assembler files (`*.asm`, `*.lst` and `*.rst`): 8051 assembler source files created by SDCC.
- ADB files (`*.adb`): Intermediate files created by SDCC containing debug information needed to create a CDB file. Note that these should not be confused with Libero ADB files and are not the same format.
- CDB files (`*.cdb`): An optional SDCC file containing debug information. Note that these should not be confused with Libero CDB files and are not the same format.
- Symbol files (`*.sym`): Symbol listing files created by the SDCC assembler.
- Linker script file (`*.lnk`): Script file used by SDCC to invoke the linker. Note that this is not the same as a GNU ld linker script file.
- Memory map file (`*.map`): SDCC memory map file for the program image. Note that this is not the same format as the map file created for ARM projects.
- Memory usage file (`*.mem`): A file summarising the 8051 memory usage. Note that this is not a mem file for use with an ENVM data storage client.

For further details about SDCC build artifacts and intermediate files refer to the SDCC manual.

Dealing with build problems

When building projects the **Console** view displays the logging output from the compilation tools as they run. If any errors or warnings arise these are also displayed in the **Problems** view from allowing them to be traced back to source or other input files.

Note that at the time of writing SDCC warnings appear in the **Problems** view as errors even if they are not fatal to the build process.

Debugging

Installing FlashPro programmer and drivers

A FlashPro programmer and connected to the target hardware before debugging can take place. Refer to the Libero/FlashPro documentation for instructions on how to install a FlashPro programmer and the required drivers and software.

By default SoftConsole uses the first FlashPro programmer found for debugging. If more than one FlashPro programmer is attached to the computer then the one that should be used for debugging must be specified to the debug sprite.

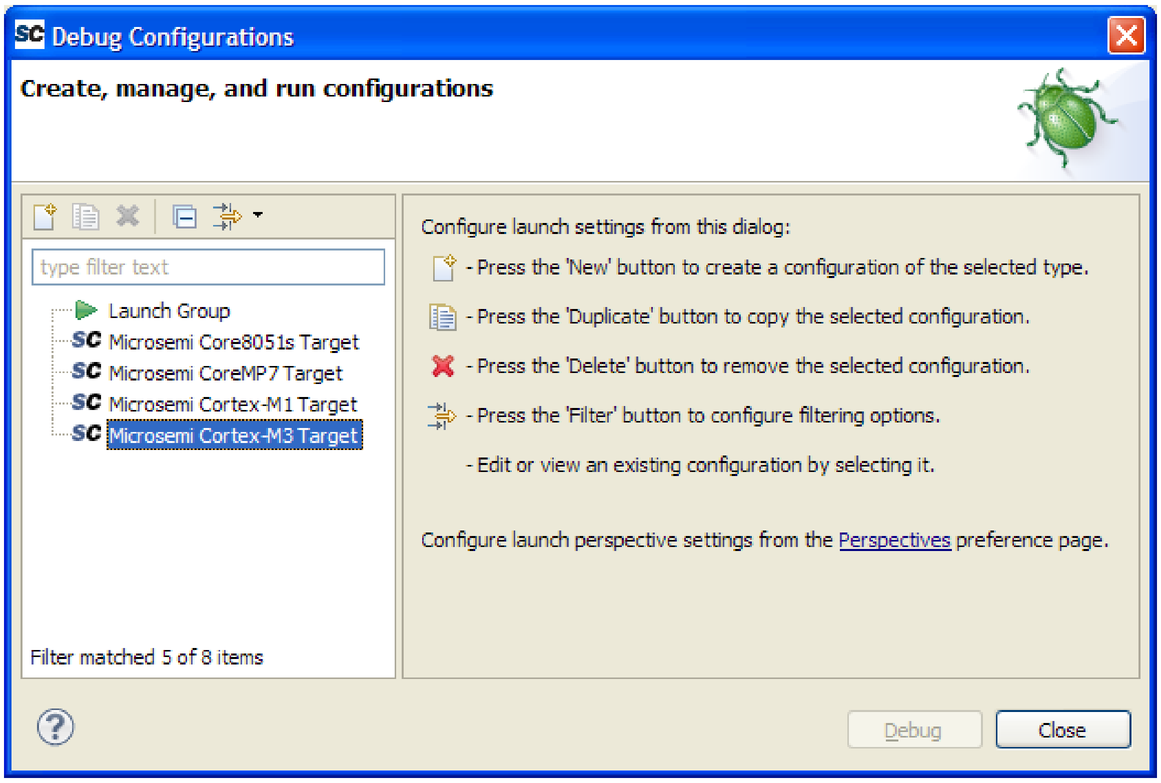
Building a project for debugging

Select a project and build configuration for debugging. By convention the **Debug** build configuration is usually the most suitable for interactive download and debugging as it normally has most or all optimization disabled and full symbolic debugging information enabled both of which facilitate accurate debugging. However it is also possible to download and debug a program that has some or all optimization enabled and some or all debugging information disabled but debugging may be more difficult – e.g. line tracking and displaying variable values may not work reliably.

Creating a debug launch configuration

To debug a build project it is necessary to create a debug launch configuration.

- Select the project in the **Project Explorer** view.
- Right click on the project and choose **Debug As... > Debug Configurations**.
- The **Debug Configurations** dialog appears:



Create, manage, and run configurations

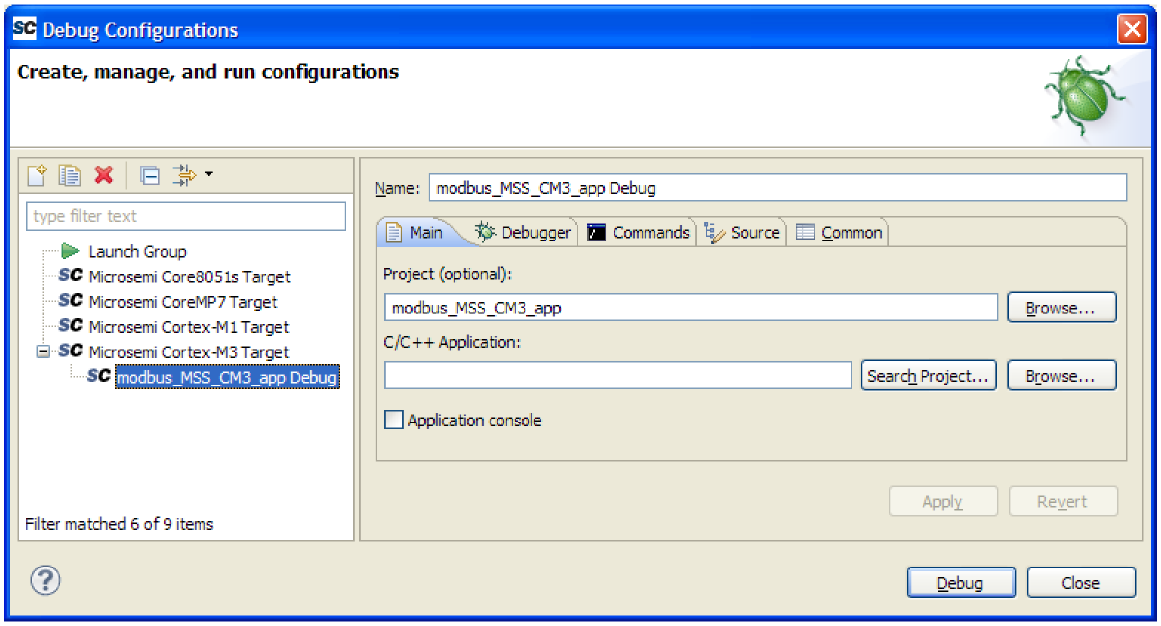
Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the Perspectives preference page.

Filter matched 5 of 8 items

- Right click on the appropriate CPU target and from the context menu choose **New**.
- The configuration settings for the debug launch configuration appear:



Debug Configurations

Create, manage, and run configurations

type filter text

- Launch Group
- Microsemi Core8051s Target
- Microsemi CoreMP7 Target
- Microsemi Cortex-M1 Target
- Microsemi Cortex-M3 Target
  - modbus_MSS_CM3_app Debug

Filter matched 6 of 9 items

Name: modbus_MSS_CM3_app Debug

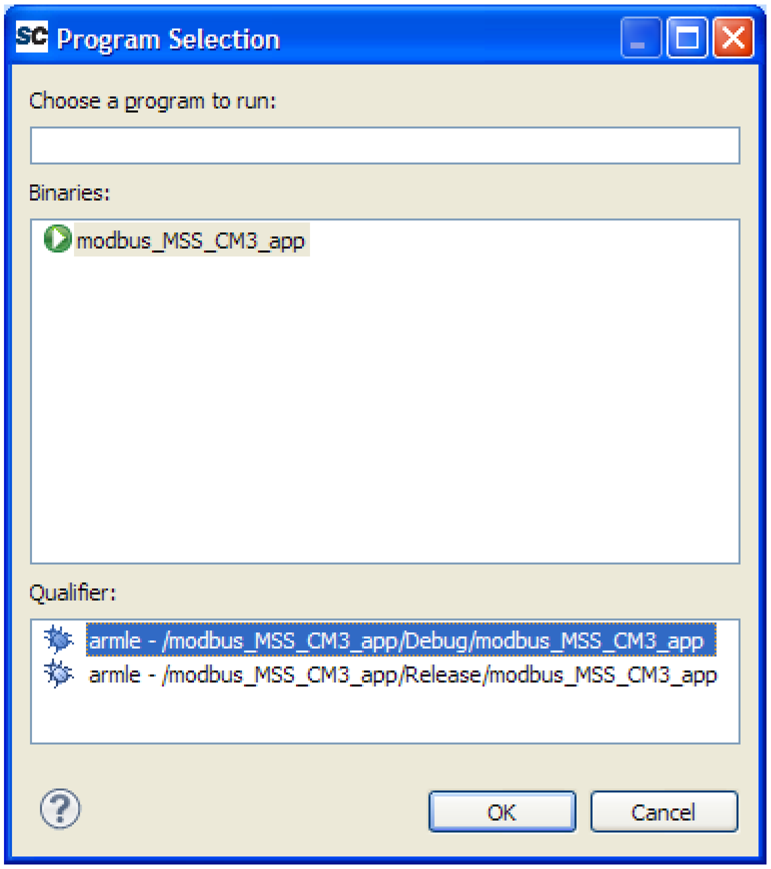
Main | Debugger | Commands | Source | Common

Project (optional):
modbus_MSS_CM3_app

C/C++ Application:

[ ] Application console

- If the **C/C++ Application** field is not automatically filled out with the executable to be debugged then click **Search Project...** and from the **Program Selection** dialog select the correct executable to be debugged and click **OK** and the **C/C++ Application** field will then be filled out.

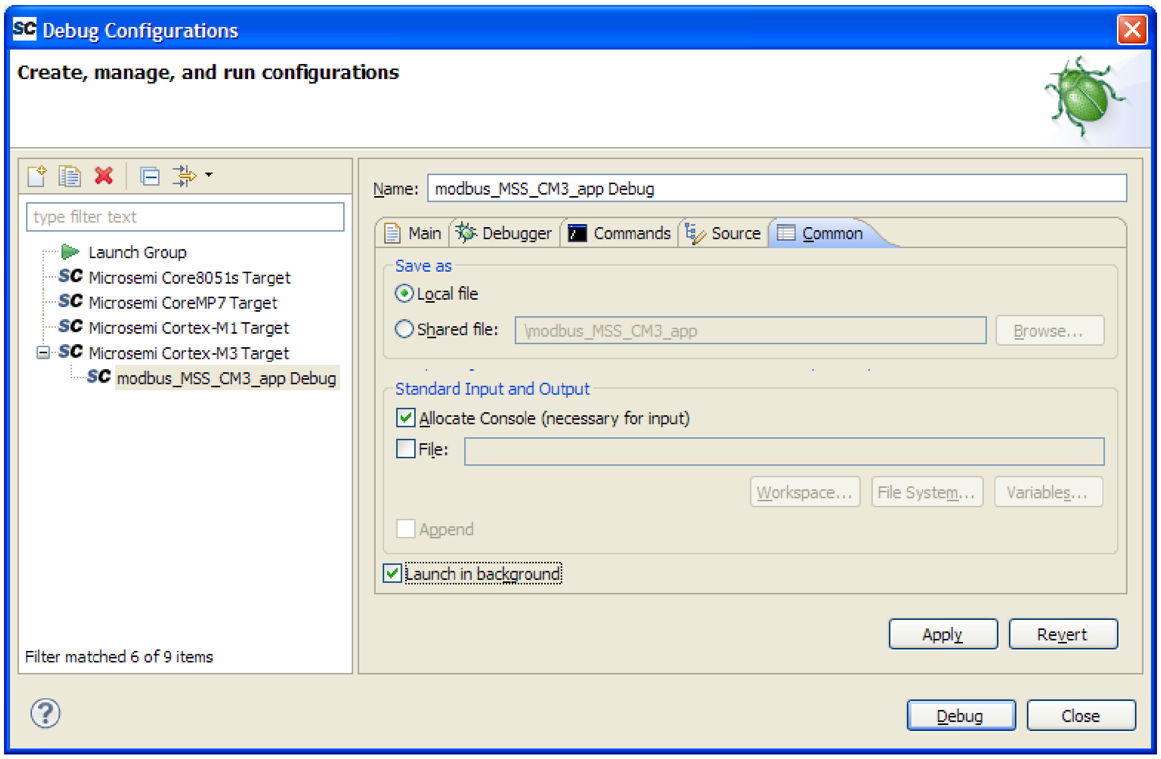


modbus_MSS_CM3_app

Qualifier:

armle - /modbus_MSS_CM3_app/Debug/modbus_MSS_CM3_app
armle - /modbus_MSS_CM3_app/Release/modbus_MSS_CM3_app

- In the **Debug Configurations** dialog click **Apply** to save the debug launch configuration settings.
- It is possible to create more than one debug launch configuration for a project if needed.
- It is often useful to configure the debug launch configuration so that the progress dialog that appears when initiating a debug session is not modal and so does not block access to the SoftConsole GUI. To do this check the **Launch in background** option on the **Common** property page:



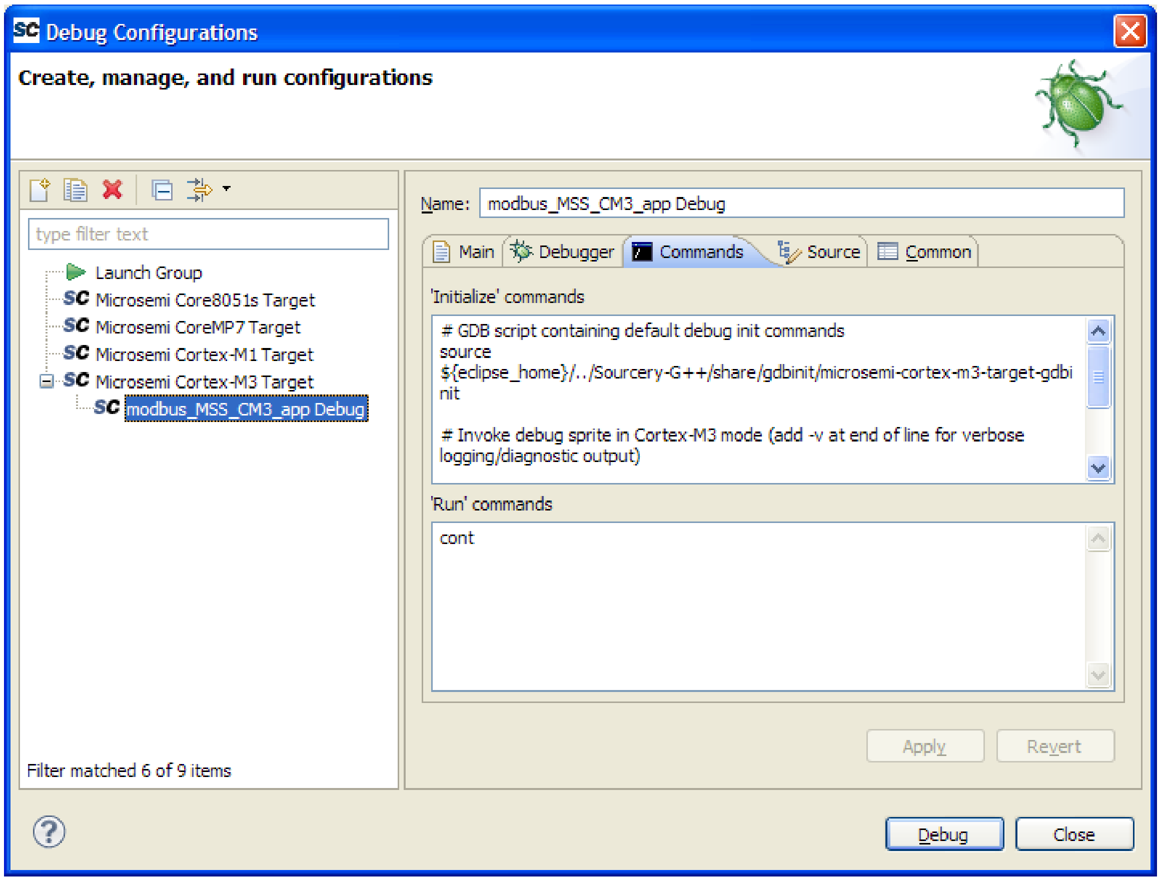
Name: modbus_MSS_CM3_app Debug

Save as
[✓ Local file]
Shared file: \modbus_MSS_CM3_app

Standard Input and Output
[✓] Allocate Console (necessary for input)
[ ] File:
[ ] Append

[✓] Launch in background

The **Commands** property page for a debug launch configuration contains some **'Initialize' commands** and some **'Run' commands** that control how GDB operates when initiating a debug session.



Name: modbus_MSS_CM3_app Debug

'Initialize' commands

```
# GDB script containing default debug init commands
source
${eclipse_home}/../Sourcery-G++/share/gdbinit/microsemi-cortex-m3-target-gdbi
nit

# Invoke debug sprite in Cortex-M3 mode (add -v at end of line for verbose
logging/diagnostic output)
```

'Run' commands

```
cont
```

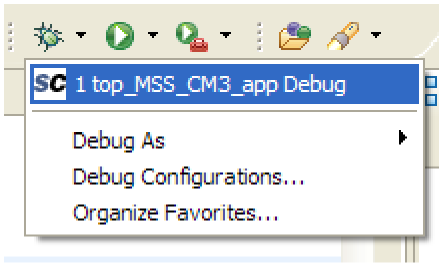
For Cortex-M3 targets (SmartFusion and SmartFusion2) most of the commands are stored in the `microsemi-cortex-m3-target-gdbinit` script file bundled with SoftConsole. For other CPU targets the commands are fully specified in the **Commands** property page.

Generally it should not be necessary to modify the `microsemi-cortex-m3-target-gdbinit` script file but if this is done then a backup copy of the original script file should be made first.

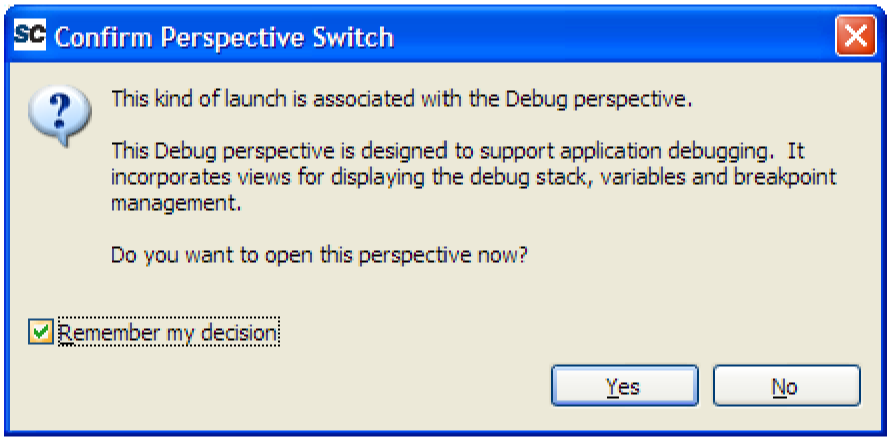
Launching a debug session

To launch a debug session invoke the relevant debug launch configuration.

- Select the project in the **Project Explorer** view.
- Right click on the project and from the context menu select **Debug As... > Debug Configurations**.
- The **Debug Configurations** dialog appears.
- Select the appropriate debug launch configuration and then click **Debug...**
- Where a debug launch configuration has been used previously it may be available to invoke from the **Debug** toolbar button menu:



- SoftConsole may prompt asking if it should switch to the **Debug** perspective – click **Yes** and optionally check the **Remember my decision** option so that it does not prompt for subsequent debug sessions.



Confirm Perspective Switch

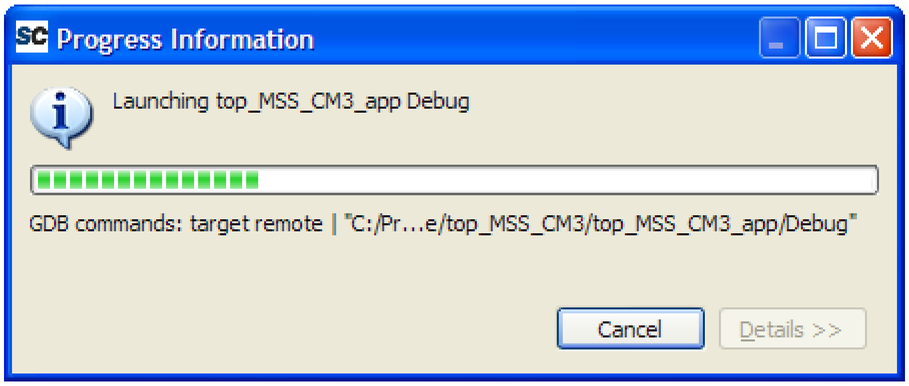
This kind of launch is associated with the Debug perspective.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

✓ Remember my decision

- The **Progress Information** dialog appears while the debug session is initiated which involves the execution of the **'Run' commands** and **'Initialize' commands** specified in the debug launch configuration:

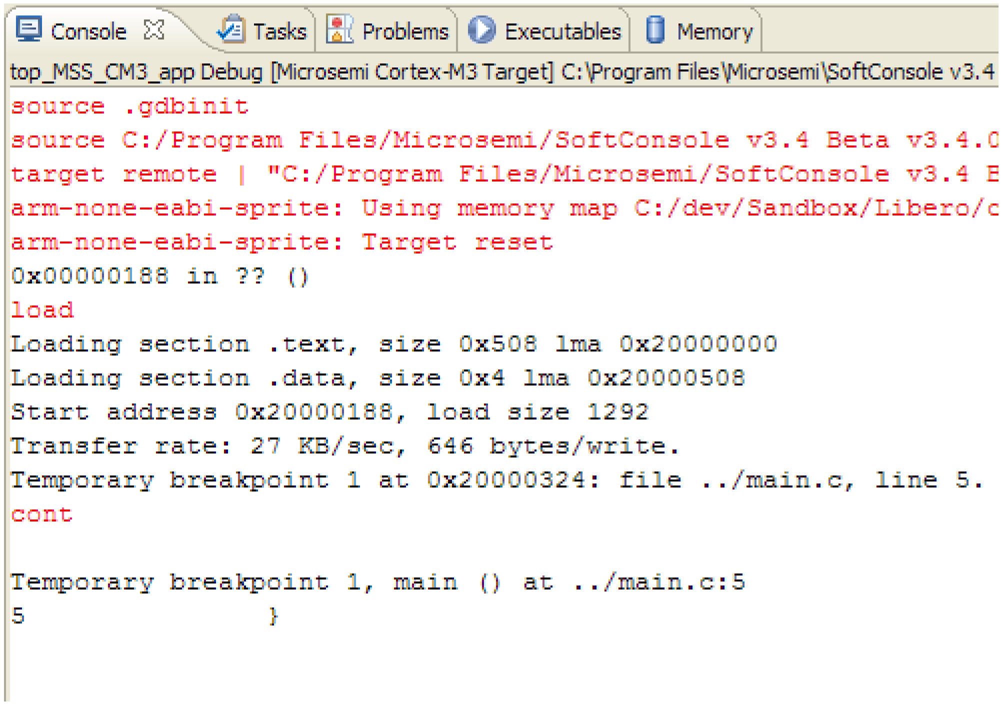


Progress Information

Launching top_MSS_CM3_app Debug

GDB commands: target remote | "C:/Pr...e/top_MSS_CM3/top_MSS_CM3_app/Debug"

- If the **Launch in background** option is enabled in the debug launch configuration then the **Progress Information** dialog will not be modal and the SoftConsole GUI can be used while it is displayed. This can be useful if there are any debug launch problems as otherwise, if the **Progress Information** dialog is modal, then clicking **Cancel** terminates the debug session altogether.
- The **Console** view will display logging and output information indicating the progress of the debug session. Note that red messages in this view are not necessarily errors!

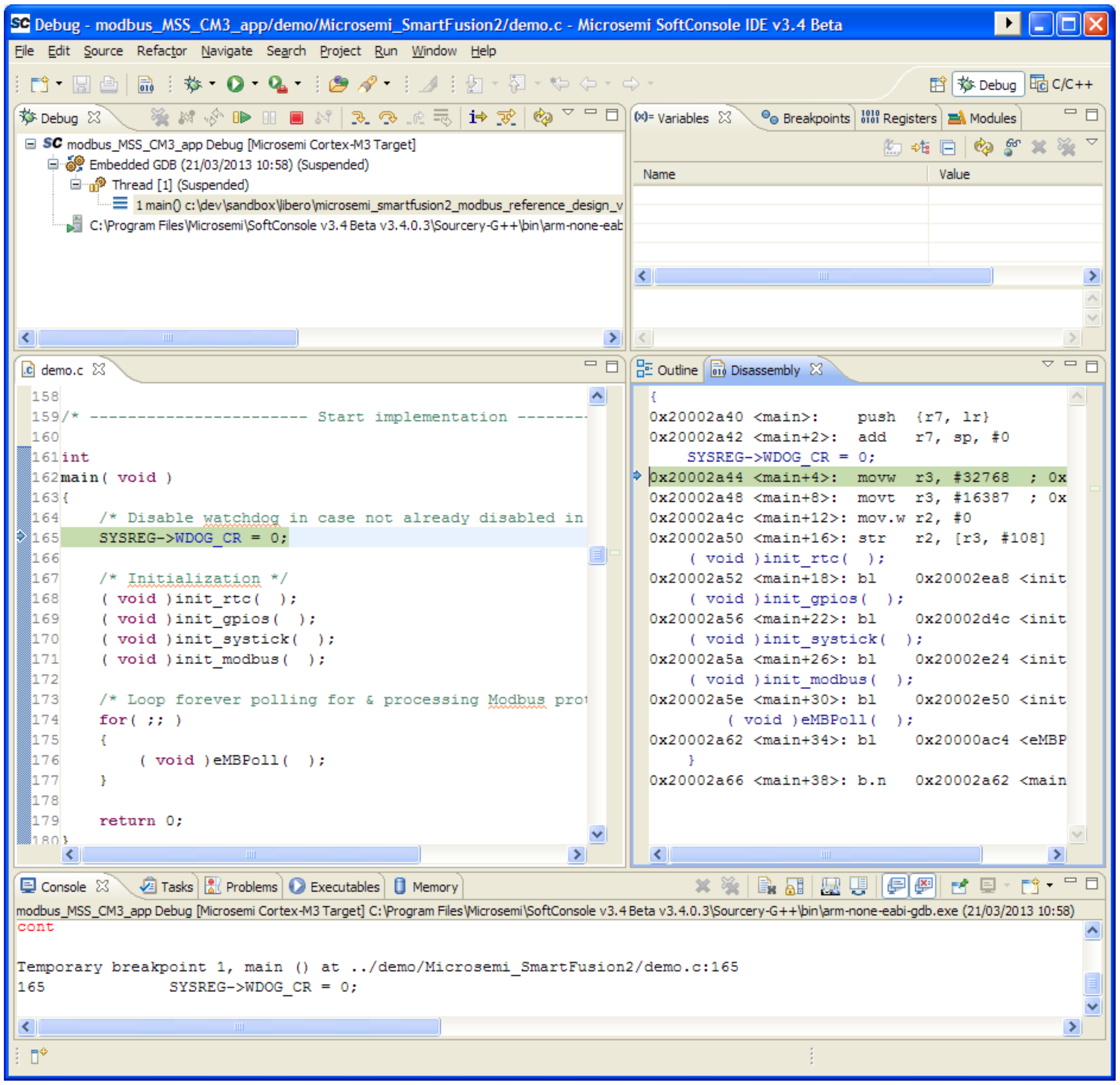


```
top_MSS_CM3_app Debug [Microsemi Cortex-M3 Target] C:\Program Files\Microsemi\SoftConsole v3.4
source .gdbinit
source C:/Program Files/Microsemi/SoftConsole v3.4 Beta v3.4.0
target remote | "C:/Program Files/Microsemi/SoftConsole v3.4 E
arm-none-eabi-sprite: Using memory map C:/dev/Sandbox/Libero/c
arm-none-eabi-sprite: Target reset
0x00000188 in ?? ()
load
Loading section .text, size 0x508 lma 0x20000000
Loading section .data, size 0x4 lma 0x20000508
Start address 0x20000188, load size 1292
Transfer rate: 27 KB/sec, 646 bytes/write.
Temporary breakpoint 1 at 0x20000324: file ../main.c, line 5.
cont

Temporary breakpoint 1, main () at ../main.c:5
5                }
```

- By default debug launch configurations will perform any necessary target initialization for debugging, establish a debug connection, download the program and then run the program until a temporary breakpoint at `main()` fires.
- Once all this has happened and SoftConsole has switched to the **Debug** perspective the stage is set for debugging the program.
- If there are any problems or errors while launching the debug session, downloading the program or executing the program then it is necessary to look at the log messages in the **Console** view (perhaps having enabled verbose debug sprite logging) and investigate the root cause of the problems.

Using a debug session

When a debug session has successfully launched and SoftConsole has switched to the **Debug** perspective it will look similar to this:


Debug - modbus_MSS_CM3_app/demo/Microsemi_SmartFusion2/demo.c - Microsemi SoftConsole IDE v3.4 Beta
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
C/C++
Debug
Variables
Breakpoints
Registers
Modules
Name
Value
demo.c
Outline
Disassembly
Console
Tasks
Problems
Executables
Memory

```
modbus_MSS_CM3_app Debug [Microsemi Cortex-M3 Target]
  Embedded GDB (21/03/2013 10:58) (Suspended)
    Thread [1] (Suspended)
      1 main() c:\dev\sandbox\libero\microsemi_smartfusion2_modbus_reference_design_v
  C:\Program Files\Microsemi\SoftConsole v3.4 Beta v3.4.0.3\Sourcery-G++\bin\arm-none-eab
```

```c

/* ----------------------- Start implementation --------

int
main( void )
{
    /* Disable watchdog in case not already disabled in
    SYSREG->WDOG_CR = 0;

    /* Initialization */
    ( void )init_rtc(  );
    ( void )init_gpios(  );
    ( void )init_systick(  );
    ( void )init_modbus(  );

    /* Loop forever polling for & processing Modbus prot
    for( ;; )
    {
        ( void )eMBPoll(  );
    }

    return 0;
}
```

```
{
0x20002a40 <main>:    push  {r7, lr}
0x20002a42 <main+2>:  add   r7, sp, #0
    SYSREG->WDOG_CR = 0;
0x20002a44 <main+4>:  movw  r3, #32768  ; 0x
0x20002a48 <main+8>:  movt  r3, #16387  ; 0x
0x20002a4c <main+12>: mov.w r2, #0
0x20002a50 <main+16>: str   r2, [r3, #108]
    ( void )init_rtc(  );
0x20002a52 <main+18>: bl    0x20002ea8 <init
    ( void )init_gpios(  );
0x20002a56 <main+22>: bl    0x20002d4c <init
    ( void )init_systick(  );
0x20002a5a <main+26>: bl    0x20002e24 <init
    ( void )init_modbus(  );
0x20002a5e <main+30>: bl    0x20002e50 <init
        ( void )eMBPoll(  );
0x20002a62 <main+34>: bl    0x20000ac4 <eMBP
    }
0x20002a66 <main+38>: b.n   0x20002a62 <main
```

```
modbus_MSS_CM3_app Debug [Microsemi Cortex-M3 Target] C:\Program Files\Microsemi\SoftConsole v3.4 Beta v3.4.0.3\Sourcery-G++\bin\arm-none-eabi-gdb.exe (21/03/2013 10:58)
cont

Temporary breakpoint 1, main () at ../demo/Microsemi_SmartFusion2/demo.c:165
165             SYSREG->WDOG_CR = 0;
```

The Eclipse and CDT help explains many of the debugging capabilities and options. Some of the commonly used views and features are explained below.

If any of the listed views is not displayed by default it can be shown by choosing **Window > Show View** from the main menu bar.

Debug view

The **Debug** view lists the active debug session and shows the stack trace. The **Debug** view has its own toolbar buttons which can be used to control debugging:

- **Resume** (continue execution), **Suspend** (pause execution) and **Terminate** (end the debug session).
- **Step Into**, **Step Over** and **Step Return** for interactive debugging of code at the C/C++ code or machine instruction level.

- **Instruction Stepping Mode** – by default this is off so that interactive debugging using **Step Into**, **Step Over** and **Step Return** operate at C/C++ code level in the relevant editor. When this mode is on debugging operates at the machine instruction level in conjunction with the code displayed in the **Disassembly** view.

When program execution is paused the **Debug** view displays the current stack trace. Clicking on any function in the stack trace shows details for that function in the editor and **Variables** view.

Editors

When a debug session is active the active editor displays the file containing the C/C++ code currently being executed. Various information about program symbols (such as functions, variables, manifest constants etc.) can be viewed by moving the mouse cursor over the relevant symbol in the editor. Breakpoints can be set or cleared by double clicking in the margin to the left hand side of the editor or by right clicking in it and using the context menu.

It is possible to edit code and save changes while a debug session is active. It is even possible to rebuild the program. However it is better not to do this as it can cause confusion while debugging when the source code or executable image no longer matches the program being debugged on the target hardware. It is better to do all code editing, saving of changes and program rebuilds in the **C/C++** perspective when no debug session is active.

Variables view

When program execution is paused the **Variables** view allows variables to be viewed and modified.

By default stack based variables in the local scope of the function currently selected in the *Debug* view stack trace are displayed. However global variables can also be added to the view by right clicking in the view and selecting **Add Global Variables...** and selecting the required variables from the **Global Variables** list.

Right clicking on a variable brings up a context menu providing additional actions such as opening a **Memory Monitor** on the area of memory in which the variable is stored, selecting the format in which the variable value is displayed etc.

Breakpoints view

The **Breakpoints** view displays all breakpoints currently configured. Because all breakpoints for all programs are displayed here it is normally a good idea to close all projects other than the project being debugged so that breakpoints for unrelated projects do not interfere with debugging.

Breakpoints can be viewed, enabled, disabled and removed in this view.

Registers view

When program execution is paused the **Registers** view displays the CPU registers. Register values can be viewed and modified from this view. Right clicking on a register and choosing **Show Memory** allows a **Memory Monitor** for the address held by the register to be created.

Outline view

The **Outline** view displays all symbols in the current source file. Clicking an item listed in the **Outline** view opens the editor at that point in the source file.

Disassembly view

The **Disassembly** view displays the program’s assembler code. It is useful for examining and debugging of code at the assembler code level. This is relevant when there is no high level C/C++ source code for a particular part of the program or when doing low level assembler level debugging using the **Debug** view’s **Instruction Stepping Mode** and program stepping options.

It is sometimes also useful to view the assembler code while debugging at the high level C/C++ language level.

As with C/C++ code in the editor it is possible to configure and enable/disable breakpoints in the **Disassembly** view using the margin displayed on the left hand side of the view.

The **Disassembly** view reads code from the target hardware and disassembles it on demand so having it visible can slow down debugging depending on the characteristics of the memory subsystem used to store program code. For this reason it is advisable to only show this view when absolutely necessary if it tends to slow debugging down.

Console view

The **Console** view shows logging and progress information for debugging and program execution. It also provides a full command line interface to GDB which means that GDB commands can be entered and executed.

Refer to the GDB documentation for details of commands supported by the debugger.

Logging and progress messages in red in the **Console** view are not necessarily errors.

Memory view

The **Memory** view displays **Memory Monitors** allowing memory regions in the target system to be displayed. Memory contents can be viewed and modified (assuming that the memory region is not read-only).

To define a **Memory Monitor** right click in the **Monitors** region of the **Memory** view and from the context menu choose **Add Memory Monitor**. In the *Monitor Memory* dialog enter the address of the memory region to view. This can also be a variable based expression.

A **Memory Monitor** can also be created from the **Variables** or **Registers** views by right clicking on a variable or register and from the context menu choosing **View Memory**.

Once a **Memory Monitor** has been defined the memory addresses and contents are displayed. Inaccessible (e.g. unimplemented or unconnected) memory regions are read as zero values and writes are ignored.

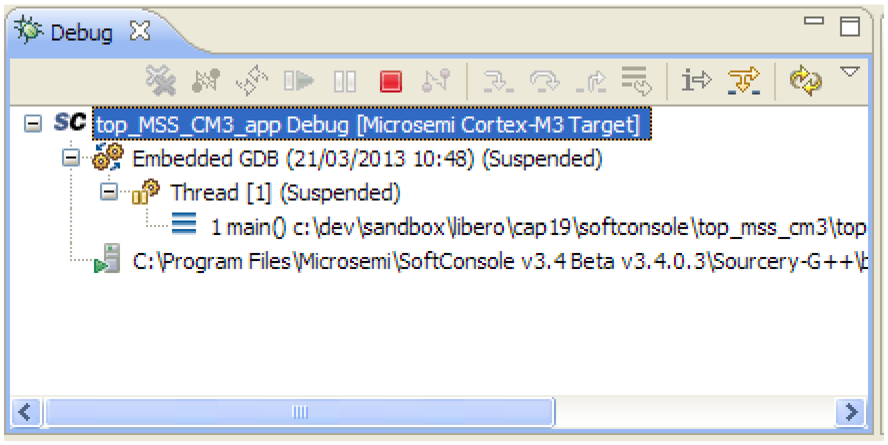
Scrolling the memory contents region allows further memory contents to be viewed. Memory contents can be changed by double clicking on a memory cell and entering a new value (assuming that memory is not read-only).

Right click on the **Memory Monitor** memory contents region and use the context menu to control characteristics such as rendering, formatting, size etc.

Right click on a **Memory Monitor** in the **Monitors** region of the **Memory** view to delete or reset it to display the memory contents at the original configured address.

Terminating a debug session

To terminate a debug session select the debug session in the **Debug** view and click the red **Terminate** button or right click on debug session and from the context menu choose **Terminate**.



```
SC top_MSS_CM3_app Debug [Microsemi Cortex-M3 Target]
   Embedded GDB (21/03/2013 10:48) (Suspended)
      Thread [1] (Suspended)
         1 main() c:\dev\sandbox\libero\cap19\softconsole\top_mss_cm3\top
   C:\Program Files\Microsemi\SoftConsole v3.4 Beta v3.4.0.3\Sourcery-G++\
```

The context menu also offers a **Terminate and Relaunch** option in order to terminate a debug session and immediately launch it again.

Once a debug session has terminated it usually makes sense to switch from the **Debug** perspective back to the **C/C++** perspective.

Flash programming

SoftConsole supports the creation of programs that can be downloaded to, debugged from and booted/executed from non volatile memory including internal ENVM on Fusion, SmartFusion and SmartFusion2 devices and external flash memory on all devices.

An obvious prerequisite for this is a CPU based system that implements a code space memory region using non volatile memory. Different systems will have different requirements and capabilities in this respect. The details of how to construct such a system are outside the scope of this document.

SoftConsole's flash programming capabilities support both ARM and 8051 based systems. The general approach is the same in both cases.

When building such a program the non volatile code space memory region and the flash programming profile to be used are specified. For ARM targets these details are provided by an annotated linker script. For 8051 targets they are provided by a memory region file passed to the `actel-map` tool.

The annotated linker script or memory region file contains a structured comment annotation indicating the non volatile code space memory region and the XML based flash programming profile to be used from this folder:

```
<SoftConsole-install-dir>\Sourcery-G++\share\sprite\flash
```

The following is an example of such an ARM linker script or memory region file annotation denoting the non volatile memory region and the XML flash programming profile to be used (note that the `.xml` file extension is omitted):

```
/* SOFTCONSOLE FLASH USE: microsemi-smartfusion2-envm */
rom (rx)   : ORIGIN = 0x60000000, LENGTH = 256k
```

The 8051 memory region file uses the same syntax as the GNU ld linker script for memory regions.

SoftConsole includes a number of useful example XML based flash programming profile files. Where these do not cater for the requirements or characteristics of a specific target system then please contact Microsemi technical support for advice on how to create a custom flash programming file.

The CMSIS/HAL firmware cores also include a number of useful example linker scripts for creating programs that can be programmed to various flash memory targets. Refer to the relevant CMSIS/HAL firmware cores and documentation for specific details about these.

When a debug session for such a program is launched the debugger knows that instead of writing the program image directly to memory (as happens with read-write memory such as RAM) it will go through the necessary flash programming steps to load the program into the non volatile memory space. This process is generally slower than downloading a program to RAM. When interactively debugging such a program the debugger is also aware that hardware rather than software breakpoints must be used.

It is also possible to create a hardware target system and build a program so that when a debug session is launched the program is programmed to the relevant target flash memory and thereafter the program boots/executes from that memory out of reset.

ARM projects

Building for flash download

As mentioned above the linker script used to link the program specifies the non volatile code space memory region and the flash programming profile to be used to download the program.

The various CMSIS/HAL firmware cores include useful example linker scripts suitable for creating programs that can be downloaded to various embedded and external flash devices. These example linker scripts refer to flash programming profiles bundled with SoftConsole.

To use one of these example linker scripts to build a program suitable for flash programming simply specify the linker script in the project build configuration properties. For example to build a Cortex-M3 project for download to and debugging from SmartFusion2 ENVM use the SmartFusion2 CMSIS HAL `debug-in-microsemi-smartfusion2-envm.ld` linker script which refers to the `microsemi-smartfusion2-envm.xml` flash programming profile which will be used for downloading the program.

The linker script for a project build configuration is specified as follows:

- Select the project in the **Project Explorer** view.
- Right click on the project and from the context menu choose **Properties**.
- Browse to **C/C++ Build > Settings > Tool Settings > GNU C Linker > Miscellaneous**
- In the **Linker flags** field enter `-T <linker-script>` where `-T` is the GCC linker script option and `<linker-script>` is the linker script itself.
- For example: `-T../CMSIS/startup_gcc/debug-in-microsemi-smartfusion2-envm.ld`

When such a program is built, a debug launch configuration created and a debug session launched the SoftConsole GDB debugger will download the program to the target flash memory using the relevant flash programming profile. The debugger displays progress information as the program is written to flash. This is done because downloading a program to flash normally takes longer than downloading to RAM. When the program has successfully downloaded debugging can proceed as normal except that hardware rather than software breakpoints are used because the code is in read-only memory.

In some cases the program downloaded in this way will also execute from the relevant flash memory out of reset.

CMSIS/HAL example linker scripts

The CMSIS/HAL firmware core linker scripts are useful examples that can be used as-is or adapted to the needs of a specific system. Refer to the CMSIS/HAL firmware cores and documentation for more detailed information about these.

Firmware core example projects and Libero generated application projects for CPU based systems use some of these example linker scripts and are useful for reference purposes.

The example linker scripts support download to and debug from embedded and external RAMs and flash devices. There are also linker scripts for creating production executables suitable for download to embedded flash using Libero/FlashPro to program the executable image to an ENVM data storage client on Fusion, SmartFusion and SmartFusion2 devices. There are also linker scripts for booting from flash and relocating to RAM.

The example linker scripts cover a variety of link/load memory map scenarios but not every possible scenario. Different systems will have different memory map, link/load, boot/run and relocate requirements. For this reason it may be necessary to adapt the CMSIS/HAL example linker scripts to the need of a specific system. Similarly it may be necessary to adapt a flash programming profile to the needs of the non volatile memory used in a specific system.

8051 projects

Specifying CODE memory in flash

By default two forms of non-volatile storage for the Core8051s are supported by SoftConsole v3.1 and higher. For each, a sample file is used by SoftConsole to properly explain the memory region’s layout to the debug sprite:

Flash Device	**Memory Region File**
Microsemi Fusion NVM	`actel-fusion-nvm-code-memory.txt`
Intel 28F640 in 8-bit mode	`intel-28f640-1x8-code-memory.txt`

Each sample file is installed in the directory

`<SoftConsole-install-dir>\src\Core8051s\memory-region-file-examples`

To make your program load into a given flash memory device, you must copy the memory region file from the install directory into your project, and then adjust a setting to make the build process use this new file.

You can add the memory region file by dragging it over from another window where you've navigated your way to the install directory. Instead, you could make SoftConsole bring the file in for you:

- Right-click your project under the **Project Explorer** tab
- Click **Import**
- Select **General > File System > Next**
- Use **Browse** to navigate your way to the `memory-region-file-examples` directory
- Select the memory region file you want to use and click **Finish**

Adjusting build for flash

After the file is in your project, right-click your project under the **Project Explorer** tab and select **Properties**. In the window that appears, select **C/C++ Build > Settings > Tool Settings > Memory map generator**.

To the existing `actel-map` command, add the option '`-M ../filename.txt`' to point it at the new file. The `actel-map` program uses this information to produce a specific memory map for that device, instead of its default description for RAM. Click **Apply** and **OK** to save your changes.

For example:

```
actel-map -M ../actel-fusion-nvm-code-memory.txt
```

produces a `memory-map.xml` file that the sprite will load and understand the program should be programmed into the Microsemi Fusion NVM device.

Debugging in flash with Core8051s

A debug session using a program running from flash memory only differs in the choice of breakpoints used by the debugger. Software breakpoints are not possible because they depend upon the ability to write a trap instruction (`0xA5`) at a given location, and flash memory is normally read-only. Instead, the Core8051s debug sprite now also supports the use of hardware breakpoints if they are available from a particular design.

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

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

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

For more information about how to construct a Core8051s base system that supports flash programming using SoftConsole please contact Micrsosemi support.

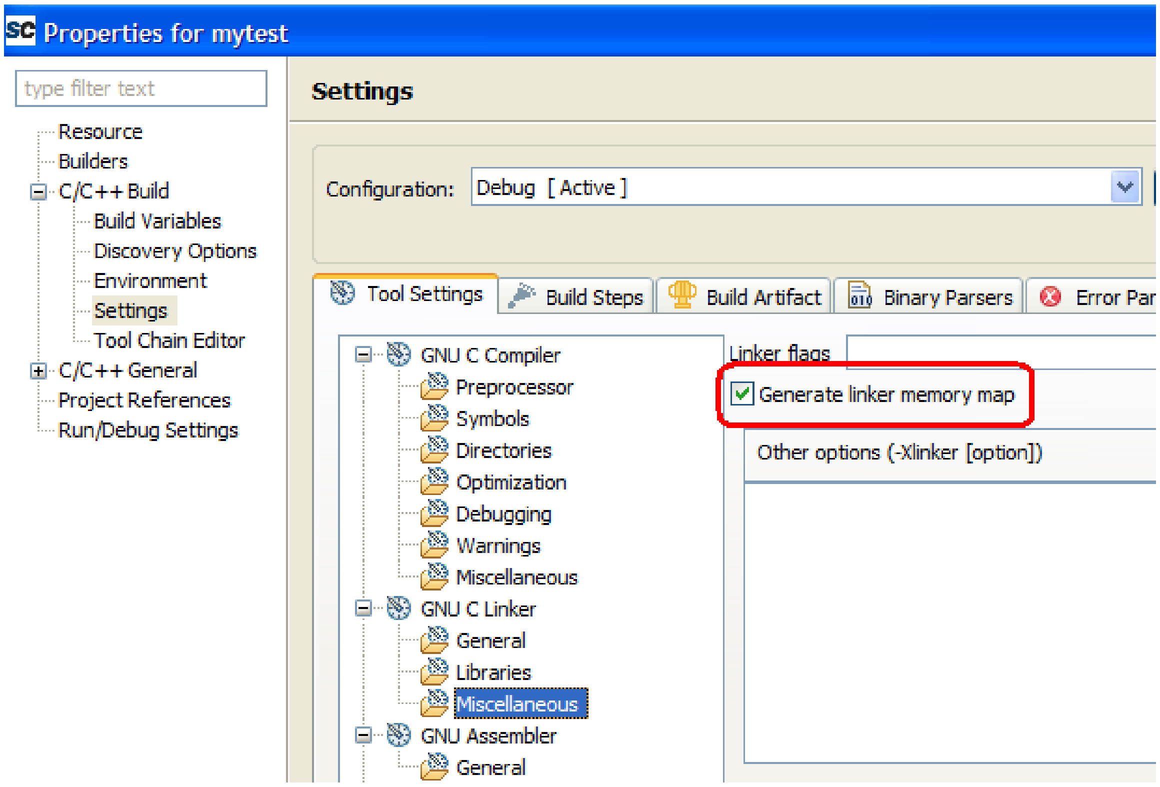
Useful tips

Linker Map files vs memory-map.xml

Previous versions of SoftConsole required you take some manual steps to create a map file with the linker; this map file can be useful when analyzing the structure and makeup of your linked application (ELF image).

Starting with SoftConsole v3.2, a linker map file is automatically generated for you as part of the standard build process. This linker map file is totally distinct from the `memory-map.xml` file automatically generated by the `actel-map` program when building a project.

Starting with SoftConsole v3.3 the linker map file generation can be controlled from a checkbox in the GUI:



Configuration: Debug [Active]

[✓] Generate linker memory map

The linker map file created during your build is used to produce information about where symbols are mapped by the linker (including whether a symbol was actually defined by the linker rather than by any particular input file). The linker map also describes allocation of storage in the resulting binary.

The debug sprite uses the `memory-map.xml` file to understand the memory regions as specified in the linker script, and explain to the debugger specific details about how to handle attempts to read or write to a given address.

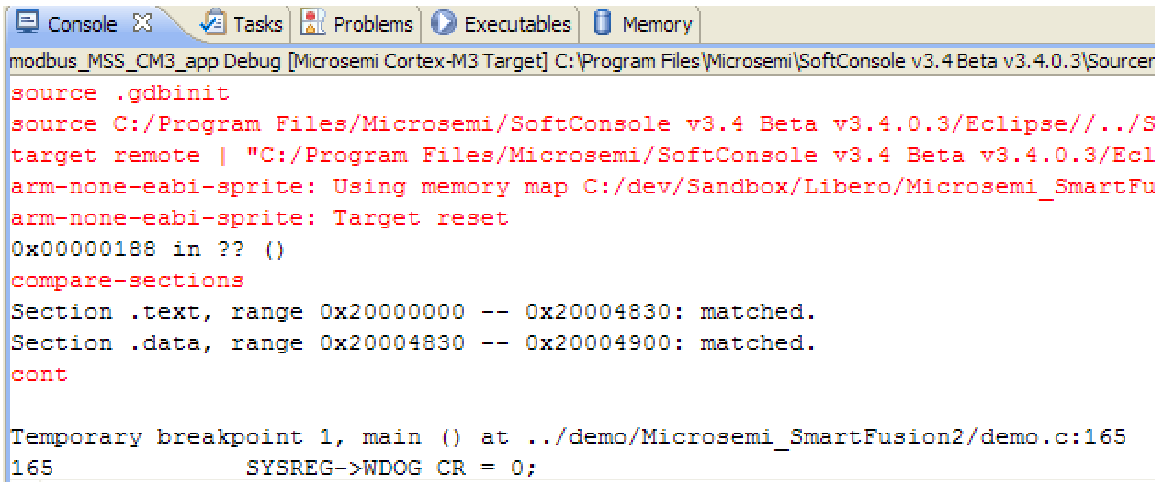
How to verify program downloads

To verify a program download:

- Select **Run > Debug Configurations...**from the main menu bar.

- Select the relevant debug launch configuration and then its **Commands** property page.
- In the **'Initialize' commands** section add the `compare-sections` command to the end of the list of commands ensuring that it appears after the `load` command.
- Save the debug launch configuration settings.

Now when the debugger is launched the `compare-sections` command will verify the integrity of the program download by comparing the contents of program memory against the program image on disk.



```
modbus_MSS_CM3_app Debug [Microsemi Cortex-M3 Target] C:\Program Files\Microsemi\SoftConsole v3.4 Beta v3.4.0.3\Sourcer
source .gdbinit
source C:/Program Files/Microsemi/SoftConsole v3.4 Beta v3.4.0.3/Eclipse//../S
target remote | "C:/Program Files/Microsemi/SoftConsole v3.4 Beta v3.4.0.3/Ecl
arm-none-eabi-sprite: Using memory map C:/dev/Sandbox/Libero/Microsemi_SmartFu
arm-none-eabi-sprite: Target reset
0x00000188 in ?? ()
compare-sections
Section .text, range 0x20000000 -- 0x20004830: matched.
Section .data, range 0x20004830 -- 0x20004900: matched.
cont

Temporary breakpoint 1, main () at ../demo/Microsemi_SmartFusion2/demo.c:165
165             SYSREG->WDOG_CR = 0;
```

How to enable verbose logging and capture logging to a file

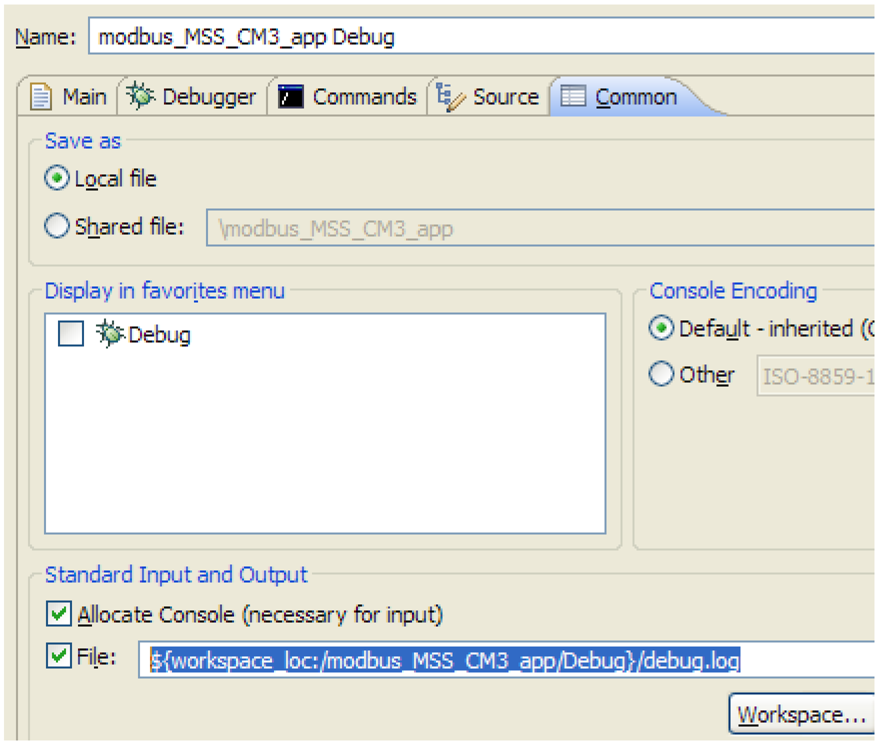
Verbose debug sprite logging can be useful when investigating problems with program download and debugging. Capturing the log output to a file can be useful for analyzing the logging information or sending it to Microsemi technical support when reporting problems.

To enable verbose logging:

- Select **Run > Debug Configurations...** from the main menu bar.
- Select the relevant debug launch configuration and then its **Commands** property page.
- In the **'Initialize' commands** section add a `-v` option to the debug sprite invocation:

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

- To capture debug logging to a file:
- Go to the **Common** property page of the selected debug launch configuration.
- Under **Standard Input and Output** check the **File** option and enter the name of a log file in the associated field. For example:


Name: modbus_MSS_CM3_app Debug

Save as
- [✓ Local file]
- Shared file: \modbus_MSS_CM3_app

Display in favorites menu
- ☐ Debug

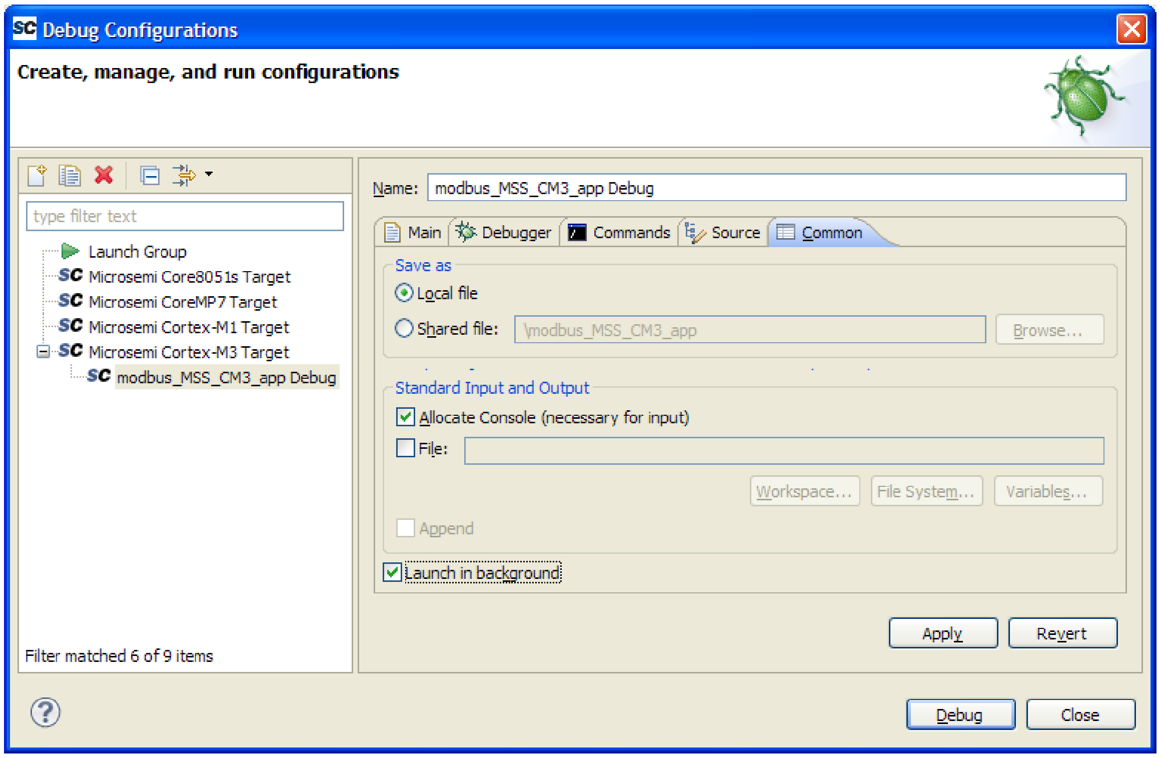
Console Encoding
- [✓ Default - inherited (]
- Other ISO-8859-1

Standard Input and Output
- ✓ Allocate Console (necessary for input)
- ✓ File: ${workspace_loc:/modbus_MSS_CM3_app/Debug}/debug.log

- Save the changes and now when the debugger is launched debug sprite log messages will be saved to the specified file.

Launch debugger in background

It is often useful to configure the debug launch configuration so that the progress dialog that appears when initiating a debug session is not modal and so does not block access to the SoftConsole GUI. To do this check the **Launch in background** option on the **Common** property page:


Debug Configurations

Create, manage, and run configurations

type filter text

- Launch Group
- Microsemi Core8051s Target
- Microsemi CoreMP7 Target
- Microsemi Cortex-M1 Target
- Microsemi Cortex-M3 Target
  - modbus_MSS_CM3_app Debug

Filter matched 6 of 9 items

Name: modbus_MSS_CM3_app Debug

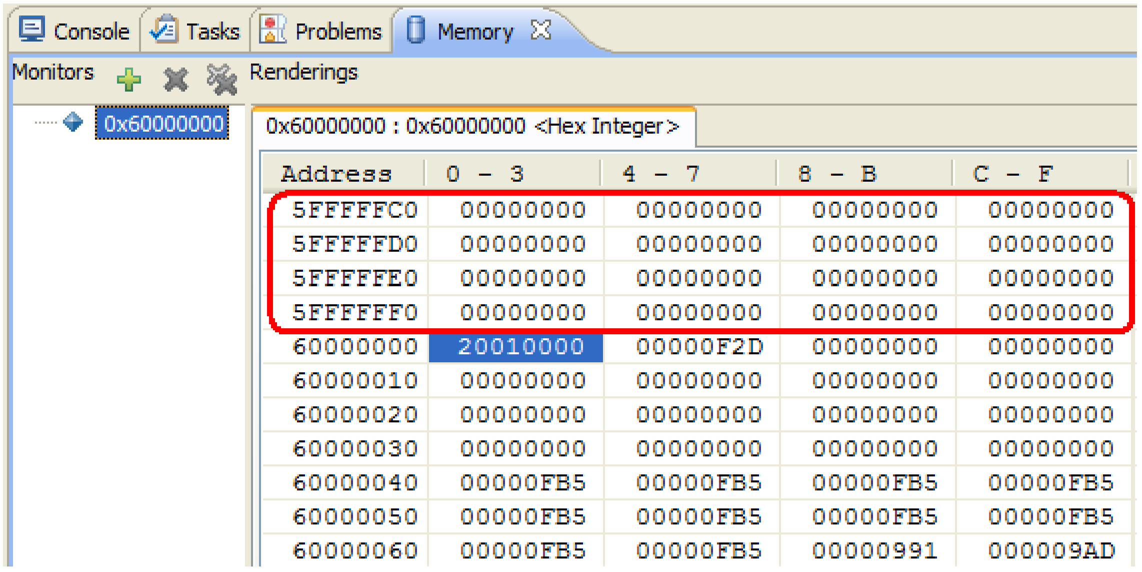
Main | Debugger | Commands | Source | Common

Save as
[✓ Local file]
Shared file: \modbus_MSS_CM3_app

Standard Input and Output
[✓] Allocate Console (necessary for input)
[ ] File:
[ ] Append
[✓] Launch in background

Access to unconnected/unimplemented memory regions

When accessing memory contents – e.g. using a **Memory Monitor** view on a specific region of memory – unconnected/unimplemented memory regions always read as zeros and writes are ignored. For example SmartFusion system registers are located at 0x60000000 and the memory region preceding this is unconnected but a **Memory Monitor** will display the following:



0x60000000 : 0x60000000 <Hex Integer>

Address	0 - 3	4 - 7	8 - B	C - F
5FFFFFC0	00000000	00000000	00000000	00000000
5FFFFFD0	00000000	00000000	00000000	00000000
5FFFFFE0	00000000	00000000	00000000	00000000
5FFFFFF0	00000000	00000000	00000000	00000000
60000000	20010000	00000F2D	00000000	00000000
60000010	00000000	00000000	00000000	00000000
60000020	00000000	00000000	00000000	00000000
60000030	00000000	00000000	00000000	00000000
60000040	00000FB5	00000FB5	00000FB5	00000FB5
60000050	00000FB5	00000FB5	00000FB5	00000FB5
60000060	00000FB5	00000FB5	00000991	000009AD

Keep only one project open at a time

To avoid confusion due to breakpoints set in one project taking effect while debugging another it is often a good idea to close all projects other than the one(s) that you are working on at the time. To do this right-click the project in the **Project Explorer** view and select **Close Unrelated Projects**.

Use “fine grained” linking to reduce executable image size

By default GCC generates an object module per compilation module (source file) and puts each object module into its own input section. The GNU ld linker then links the whole of an input section if any function or data symbol that it contains is referenced. This can lead to linked executable images that are larger than absolutely necessary. For example if an input section contains three different symbols only one of which is actually referenced then all three will be linked into the resulting output image.

“Fine grained” linking can be used to circumvent this issue without having to otherwise change the way that programs are structured and compiled. The GCC options `-ffunction-sections` and `-fdata-sections` put all function and data symbols respectively into their own separate input sections. The GNU ld linker option `-gc-sections` “garbage collects” (omits) unused input sections. In the previous example if these options were used to build the program each of the three symbols would have its own input section and only those symbols actually referenced would be linked into the resulting executable output image.

To use these options:

- Under **Properties > C/C++ Build > Settings > GNU C[++] Compiler > Optimization > Other optimization** flags add `-ffunction-sections -fdata-sections`
- Under **Properties > C/C++ Build > Settings > GNU C[++] Linker > Miscellaneous > Linker flags** add `-gc-sections`
- If one or more library projects are used in conjunction with an application project then repeat the previous steps for each library project.
- Perform a clean build of the project.

One caveat that applies is that so called “magic sections” that are required in the output image but are not explicitly referenced by symbol will be omitted when using these compiler/linker options.

Avoid unnecessary use `float/double`

Only use floating point variables and arithmetic when absolutely necessary as doing so can cause a significant amount of floating point support library overhead to be linked in. In many cases integer only arithmetic will suffice and will yield significantly smaller executable images.

newlib integer only `*printf()/*scanf()` support

If you choose to use newlib for formatted input/output then bear in mind that it supports integer only versions of the `*printf()/*scanf()` methods which are lighter weight than the full blown (including float support) versions and which may be more suitable for use on an embedded platform. For more information refer to the newlib documentation at http://sourceware.org/newlib/ – in particular the documentation on `viprintf()` and `viscanf()`.

Remember that to use newlib for such input/output you will still need to retarget the relevant newlib Syscalls (e.g. `write()`, `write_r()`, `read()`, `read_r()`) to route input/output via your device.

The SmartFusion CMSIS-PAL and SmartFusion2 CMSIS Hardware Abstraction Layer firmware cores support redirection of newlib `*printf()` output to an MSS UART/MMUART through the use of specific `#define` manifest constants outlined in the `CMSIS/startup_gcc/newlib_stubs.c` source file. Refer to the CMSIS-PAL documentation for more information. Refer to the relevant CMSIS documentation and sample projects for more information.

Use a lightweight `[s]printf()` implementation

In many cases only basic `[s]printf()` support is required for embedded applications – e.g. integer only and/or only a limited set of formatting specifiers and options. As mentioned above the integer only `*printf()` support in Newlib can be useful in this context and reduce the size of the linked executable image compared to one that uses the “full blown” `*printf()` library support. In some cases even more limited `[s]printf()` support will suffice in which case an even smaller implementation can be used. For example:

- Kustaa Nyholm’s tiny `[s]printf()` licensed under GPL:
  - Initial version: http://www.sparetimelabs.com/tinyprintf/index.html
  - Subsequent smaller and more limited (e.g. no `sprintf()` support) version: http://www.sparetimelabs.com/printfrevisited/index.html

- georges@menie.org’s small `printf()` licensed under LGPL:
  - http://www.menie.org/georges/embedded/index.html#printf

Licensing

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

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

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

Visit the Microsemi SoC Products Group Customer Support website for more information and support (http://www.microsemi.com/soc/support/search/default.aspx). Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on website.

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. Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

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 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

XXXXXXX-X/MM.YY