SmartFusion2 CMSIS Hardware Abstraction Layer User's Guide

Version 2.3

Microsemi.

Table of Contents

Introduction

The SmartFusion2™ CMSIS Hardware Abstraction Layer (CMSIS-HAL) provides boot code and hardware access methods for the SmartFusion2 Cortex-M3 microprocessor subsystem (MSS).

Features

The SmartFusion2 CMSIS-HAL provides support for the following features:

- Cortex-M3 boot code for the SoftConsole, Keil-MDK and IAR Embedded Workbench tool chains
- SmartFusion2 MSS peripherals register access data structures and register read/write macros
- Automatic configuration of the SmartFusion2 MDDR and FDDR memory controllers based on the configuration selected in the Libero hardware flow
- Automatic configuration of the SmartFusion2 SERDES hardware blocks based on the configuration selected in the Libero hardware flow
- Hardware abstraction layer for Microsemi soft IP drivers
- Optional redirection of the standard C library output to one of the SmartFusion2 MSS MMUART

The CMSIS-HAL is provided as C and assembly source code.

Supported Hardware IP

The SmartFusion2 CMSIS Hardware Abstraction Layer can be used with the SmartFusion2 MSS version 0.0.500 or higher.

Files Provided

The files provided as part of SmartFusion2 CMSIS-HAL fall into three main categories: documentation, core source code (including linker setup) and debug configuration files. The CMSIS-HAL is distributed via the Microsemi SoC Products Group's Firmware Catalog, which provides access to the documentation for the CMSIS-HAL, generates the CMSIS-HAL source files for the selected tool chain into an application project. and generates debug configuration files for the selected tool chain. The Firmware Catalog is available from: www.microsemi.com/soc/products/software/firmwarecat/default.aspx.

Documentation

The Firmware Catalog provides access to these documents for the driver:

- User's guide (this document)
- Release notes

Core CMSIS Source Code

The Firmware Catalog generates the CMSIS-HAL source code into the *CMSIS* subdirectory of the selected software project directory. For example, <*my_project*>*/CMSIS*. The files making up the CMSIS are detailed below.

m2sxxx.h

This header file contains the SmartFusion2 MSS interrupt numbers, MSS peripheral memory map and MSS peripheral register descriptions. You can include this file in application code directly accessing MSS hardware registers.

system_m2sxxx.h

This header file contains the prototype for the *SystemCoreClockUpdate()* function and the global variables used to communicate the frequency of the various SmartFusion2 clocks. The clock frequency global variables are initialized to the Libero hardware design values defined in the *sys_config_mss_clocks.h* file in the *drivers_config/sys_config* directory. The *SystemCoreClockUpdate()* function can be used in application code to ensure the clock frequency global variables are up to date. You can include this header file in application code requiring knowledge of some of the system's clock frequencies.

system_m2sxxx.c

This C source file contains the implementation of the *SystemInit()* function called as part of the boot code to configure the SmartFusion2 system. This file must always be part of your software project.

mss_assert.h

This header file contains assertion macros used by the Microsemi SmartFusion2 MSS peripheral drivers. You can included this file in application if you want to make use of the *ASSERT()* assertion macro.

sys_init_cfg_types.h

This header file contains data structure definitions used to retrieve configuration data generated as part of the Libero hardware flow. This file should not be included in application code.

hw_reg_io.h

This header file contains register access macros used by some of the Microsemi SmartFusion2 MSS peripheral drivers. This file should not be included in application code.

SoftConsole Specific Source Code

These files are located in the directory <*my_project*>*/CMSIS/startup_gcc*.

startup_m2sxxx.s

This assembly code file contains the Cortex-M3 vector table and default exception handlers.

newlib_stubs.c

This C source file contains stubs for the Newlib system calls. It is only use in SoftConsole projects.

debug-in-microsemi-smartfusion2-external-ram.ld

This GNU linker script is an example linker script intended for use with SoftConsole. It can be used as a starting point for applications debugged with SoftConsole where the executable is located in external DDR memory.

debug-in-microsemi-smartfusion2-envm.ld

This GNU linker script is an example linker script intended for use with SoftConsole. It can be used as a starting point for applications debugged with SoftConsole where the executable is located in internal eNVM.

debug-in-microsemi-smartfusion2-esram.ld

This GNU linker script is an example linker script intended for use with SoftConsole. It can be used as a starting point for applications debugged with SoftConsole where the executable is located in internal eSRAM.

production-smartfusion2-execute-in-place.ld

This GNU linker script is an example linker script intended for use with SoftConsole. It can be used as a starting point for generating an executable image that will be loaded in eNVM using the Libero hardware flow or the SoftConsole debugger flash loader. This linker script is for executing the application code directly from eNVM.

production-smartfusion2-relocate-to-external-ram.ld

This GNU linker script is an example linker script intended for use with SoftConsole. It can be used as a starting point for generating an executable image that will be loaded in eNVM using the Libero hardware flow or the SoftConsole debugger flash loader. This linker script is for booting from eNVM then copying and running the executable in DDR memory.

IAR Embedded Workbench Specific Source Code

These files are located in the directory *<my_project>/CMSIS/startup_iar*.

startup_m2sxxx.s

This assembly code file contains the Cortex-M3 reset handler.

write.c

This C source file is only used with the IAR Embedded Workbench tool chain. It handles the redirection of standard library I/O to one of the SmartFusion2 MSS UART. This file only needs to be included in an IAR Embedded Workbench project if you wish to redirect the output of *printf()* to one of the serial ports.

read.c

This C source file is only used with the IAR Embedded Workbench tool chain. It handles the redirection of standard library I/O to one of the SmartFusion2 MSS UART. This file only needs to be included in an IAR Embedded Workbench project if you wish to use one of the serial ports as input for the *scanf()* function.

low_level_init.c

This C source file is used with the IAR Embedded Workbench tool chain. It handles the memory remapping and setting the Cortex-M3 vector table offset register.

default_irqs.c

This C source file is used with the IAR Embedded Workbench tool chain. It contains the default implementation of the exception handlers and interrupts service routines.

vector_table.c

This C source file is used with the IAR Embedded Workbench tool chain. It contains the Cortex-M3 vector table.

smartfusion2_esram_debug.icf

This is an example linker configuration file intended for use with IAR Embedded Workbench. It can be used as a starting point for applications debugged with IAR Embedded Workbench where the executable is located in external RAM.

smartfusion2_envm.icf

This is an example linker configuration file intended for use with IAR Embedded Workbench. It can be used as a starting point for applications debugged with IAR Embedded Workbench where the executable is located in internal eNVM.

smartfusion2_mddr_debug.icf

This is an example linker configuration file intended for use with IAR Embedded Workbench. It can be used as a starting point for applications debugged with IAR Embedded Workbench where the executable is located in external DDR memory.

smartfusion2_envm_to_mddr.icf

This is an example linker configuration file intended for use with IAR Embedded Workbench. It can be used as a starting point for generating an executable image that will be loaded in eNVM using the Libero hardware flow or the IAR Embedded Workbench debugger flash loader. This linker configuration file is for booting from eNVM and copying and running the executable in DDR memory.

Keil-MDK Specific Source Code

These files are located in the directory *<my_project>/CMSIS/startup_arm*.

startup_m2sxxx.s

This assembly code file contains the Cortex-M3 vector table and default exception handlers.

retarget.c

This C source file is only used with the Keil-MDK tool chain. It handles the redirection of standard library I/O to one of the SmartFusion2 MSS UART. This file only needs to be included in a Keil-MDK project if you wish to redirect the output of *printf()* to one of the serial ports.

low_level_init.c

This C source file is used with the Keil-MDK tool chain. It handles the memory remapping and setting the Cortex-M3 vector table offset register.

smartfusion2_esram_debug.sct

This is an example linker scatter file intended for use with Keil-MDK. It can be used as a starting point for applications debugged with Keil-MDK where the executable is located in external RAM.

smartfusion2_execute_in_place.sct

This is an example linker scatter file intended for use with Keil-MDK. It can be used as a starting point for applications debugged with Keil-MDK where the executable is located in internal eNVM.

smartfusion2_mddr_debug.sct

This is an example linker scatter file intended for use with Keil-MDK. It can be used as a starting point for applications debugged with Keil-MDK where the executable is located in external DDR memory.

smartfusion2_relocate_to_external_ram.sct

This is an example linker scatter file intended for use with Keil-MDK. It can be used as a starting point for generating an executable image that will be loaded in eNVM using the Libero hardware flow or the Keil-MDK debugger flash loader. This linker scatter file is for booting from eNVM and copying and running the executable in DDR memory.

Libero Generated Hardware Configuration Source Code

These files are located in the directory <*my_project*>*/drivers_config/sys_config*.

The Libero hardware design flow generates the hardware configuration source code for the CMSIS-HAL into the *firmware/drivers_config/sys_config* subdirectory of the Libero project.

The *firmware/drivers_config/sys_config* directory must be copied to *drivers_config/sys_config* of the selected software project directory. If this folder is already present in the software project, it should be deleted before copying the Libero generated folder.

sys_config.h

This header file contains information about the SmartFusion2 MSS hardware configuration. It is generated by the Libero hardware design flow. The content of this file is hardware design specific. This file should not be included in application code.

sys_config.c

This C source file contains information about the SmartFusion2 MSS hardware configuration. It is generated by the Libero hardware design flow. The content of this file is hardware design specific. This file must be part of your software project if the hardware design uses one of the DDR memory controllers or a SERDES interface.

sys_config_mss_clocks.h

This header file contains information about the SmartFusion2 MSS hardware clock configuration. It is generated by the Libero hardware design flow. The content of this file is hardware design specific. This file should not be included in application code.

sys_config_mddr_define.h

This header file contains information about the SmartFusion2 MSS DDR hardware configuration. It is generated by the Libero hardware design flow if DDR is included in the Libero design. The content of this file is hardware design specific. This file should not be included in application code.

sys_config_SERDESIF_<0-3>.c

These C source files contain information about the SmartFusion2 SERDES interface hardware configuration. They are generated by the Libero hardware design flow if SERDES interfaces are included in the Libero design. A separate file is generated for each SERDES interface. The content of these files is hardware design specific. These files must be part of your software project if the hardware design uses one or more SERDES interfaces.

sys_config_SERDESIF_<0-3>.h

These header files contain information about the SmartFusion2 SERDES interface hardware configuration. They are generated by the Libero hardware design flow if SERDES interfaces are included in the Libero design. A separate file is generated for each SERDES interface. The content of these files is hardware design specific. These files should not be included in application code.

Soft IP Bare Metal Drivers Hardware Abstraction Layer

These files are located in the directory <*my_project*>*/hal*.

The Firmware Catalog generates the CMSIS Soft IP Bare Metal Hardware Abstraction Layer source code into the *hal* subdirectory of the selected software project directory. These file are used in the implementation of bare metal drivers for hardware soft IP. The application code does not need to include any of the header files in this subdirectory as they are only used by Microsemi provided bare metal drivers.

Files *hal.s* and *hw_reg_access.s* must be part of your software project if you are using soft IP drivers.

Soft IP Bare Metal Drivers

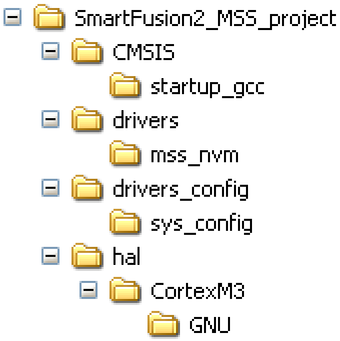
These files are located in the directory *<my_project>/driver*.

The Firmware Catalog or the Libero hardware design flow generates the DirectCore soft IP drivers into subdirectories in this folder. Any C files present should be included as part of your project.

CMSIS-HAL Deployment

The CMSIS-HAL is deployed from the Firmware Catalog into a software project by generating the CMSIS-HAL source files into the project directory.

The following example shows the intended directory structure for a SoftConsole ARM® Cortex™-M3 project targeted at the SmartFusion2 MSS. This project uses the MSS eNVM driver. This driver relies on SmartFusion2 CMSIS-HAL for accessing the hardware. The contents of the *drivers* directory result from generating the source files for the drivers used into the project from the list of drivers in the Firmware Catalog. The contents of the *CMSIS* and *hal* directories result from generating the source files for the SmartFusion2 CMSIS-HAL into the project. The contents of the *drivers_config* directory are generated by the Libero project and must be copied into the into the software project.



```
SmartFusion2_MSS_project
    CMSIS
        startup_gcc
    drivers
        mss_nvm
    drivers_config
        sys_config
    hal
        CortexM3
            GNU
```

Figure 1 · SmartFusion2 MSS Project Example

CMSIS-HAL Configuration

The SmartFusion2 CMSIS-HAL provides infrastructure code handling the SmartFusion2 processor boot sequence and system configuration. It relies on hardware configuration data generated by the Libero hardware flow. The SmartFusion2 CMSIS-HAL is not intended to be manually configured.

The CMSIS-HAL consumes hardware configuration data in the form of C header files and data structures. These headers files and configuration data C source code are generated by Libero into the *drivers_config/sys_config* folder of the exported firmware. These Libero generated source code files must be copied into the embedded software project for SmartFusion2 to operate correctly. You must ensure that these files are in synch between the Libero design and embedded software project. Doing otherwise may result in unexpected results.

CMSIS-HAL Features

Interrupt Service Routines

The SmartFusion2 CMSIS-HAL provides basic infrastructure for handling exceptions and interrupts. It contains the Cortex-M3 interrupt vector table and default interrupt service routines. These default exception and interrupt service routines provided by the CMSIS-HAL can be overridden by the application by providing functions with predefined names. These function names are listed in the sections following.

Processor Exceptions

The table below lists the function names that can be used by the application to provide a handler for a specific Cortex-M3 exception.

Exception Number	Handler Function	Description
`NonMaskableInt_IRQn`	`NMI_Handler()`	Watchdog interrupt
`HardFault_IRQn`	`HardFault_Handler()`	Hard Fault
`MemoryManagement_IRQn`	`MemManage_Handler()`	Memory Management Fault
`BusFault_IRQn`	`BusFault_Handler()`	Bus error
`UsageFault_IRQn`	`UsageFault_Handler()`	Program error exception
`SVCall_IRQn`	`SVC_Handler()`	System Service call
`DebugMonitor_IRQn`	`DebugMon_Handler()`	Debug Monitor
`PendSV_IRQn`	`PendSV_Handler()`	Pendable system service request
`SysTick_IRQn`	`SysTick_Handler()`	System Tick Timer interrupt

Table 1 · Standard Cortex-M3 Exceptions

Peripheral Interrupts

Interrupts can be controlled using the standard ARM CMSIS NVIC interrupt control functions. Interrupt sources are identified to these functions using the NVIC interrupt number.

```
void NVIC_EnableIRQ(IRQn_Type IRQn);
void NVIC_DisableIRQ(IRQn_Type IRQn);
void NVIC_ClearPendingIRQ(IRQn_Type IRQn);
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority);
```

The table below summarizes the NVIC interrupt numbers and the name of the handler function for each interrupt source. Adding handling of a particular exception in your application is as simple as creating a function using one of these predefined exception handler names. The name of these exception handler functions is defined as part of the CMSIS-HAL startup code. The default implementation provided as part of the CMSIS-HAL startup code is defined with weak linking meaning that any handler function of the same name defined as part of your implementation will override the default implementation.

Interrupt Number	Handler Function	Interrupt Source
`WdogWakeup_IRQn`	`WdogWakeup_IRQHandler()`	Watchdog
`RTC_Wakeup_IRQn`	`RTC_Wakeup_IRQHandler()`	Real Time Counter
`SPI0_IRQn`	`SPI0_IRQHandler()`	SPI 0

Interrupt Number	Handler Function	Interrupt Source
SPI1_IRQn	SPI1_IRQHandler()	SPI 1
I2C0_IRQn	I2C0_IRQHandler()	I2C 0
I2C0_SMBAlert_IRQn	I2C0_SMBAlert_IRQHandler()	I2C 0
I2C0_SMBus_IRQn	I2C0_SMBus_IRQHandler()	I2C 0
I2C1_IRQn	I2C1_IRQHandler()	I2C 1
I2C1_SMBAlert_IRQn	I2C1_SMBAlert_IRQHandler()	I2C 1
I2C1_SMBus_IRQn	I2C1_SMBus_IRQHandler()	I2C 1
UART0_IRQn	UART0_IRQHandler()	MMUART 0
UART1_IRQn	UART1_IRQHandler()	MMUART 1
EthernetMAC_IRQn	EthernetMAC_IRQHandler()	Ethernet MAC
DMA_IRQn	DMA_IRQHandler()	Peripheral DMA block
Timer1_IRQn	Timer1_IRQHandler()	Timer 1
Timer2_IRQn	Timer2_IRQHandler()	Timer 1
CAN_IRQn	CAN_IRQHandler()	CAN Controller
ENVM0_IRQn	ENVM0_IRQHandler()	eNVM Controller
ENVM1_IRQn	ENVM1_IRQHandler()	eNVM Controller
ComBlk_IRQn	ComBlk_IRQHandler()	System Controller Comm Block
USB_IRQn	USB_IRQHandler()	USB
USB_DMA_IRQn	USB_DMA_IRQHandler()	USB
PLL_Lock_IRQn	PLL_Lock_IRQHandler()	MSSDDR PLL
PLL_LockLost_IRQn	PLL_LockLost_IRQHandler()	MSSDDR PLL
CommSwitchError_IR	CommSwitchError_IRQHandler()	Communication Switch
CacheError_IRQn	CacheError_IRQHandler()	Cache Controller
DDR_IRQn	DDR_IRQHandler()	MDDR Memory Controller
HPDMA_Complete_IRQn	HPDMA_Complete_IRQHandler()	High Speed DMA Controller
HPDMA_Error_IRQn	HPDMA_Error_IRQHandler()	High Speed DMA Controller
ECC_Error_IRQn	ECC_Error_IRQHandler()	MDDR Error Correction Controller
MDDR_IOCalib_IRQn	MDDR_IOCalib_IRQHandler()	MDDR Memory Controller
FAB_PLL_Lock_IRQn	FAB_PLL_Lock_IRQHandler()	PLL
FAB_PLL_LockLost_IRQn	FAB_PLL_LockLost_IRQHandler()	PLL
FIC64_IRQn	FIC64_IRQHandler()	Fabric Interface
FabricIrq<n>_IRQn	FabricIrq<n>_IRQHandler()	Fabric Interface. <n> is 0 to 15.
GPIO<n>_IRQn	GPIO<n>_IRQHandler()	MSS GPIOs. <n> is 0 to 31

Table 2 · Peripheral Interrupts Summary

System clocks frequencies global variables

The SmartFusion2 CMSIS-HAL provides a set of global variables for communicating the frequency of the various SmartFusion2 MSS clocks. These variables are initialized to the values that are defined in the sys_config_mss_clocks.h file in the *drivers_config\sys_config* directory, which must first be copied from the *firmware/drivers_config/sys_config* subdirectory of the Libero project.

SystemCoreClock

The *SystemCoreClock* global variable contains the value of the Cortex-M3 core clock frequency.

g_FrequencyPCLK0

The *g_FrequencyPCLK0* global variable contains the value of the APB 0 bus clock frequency.

g_FrequencyPCLK1

The *g_FrequencyPCLK1* global variable contains the value of the APB 1 bus clock frequency.

g_FrequencyPCLK2

The *g_FrequencyPCLK2* global variable contains the value of the APB 2 bus clock frequency.

g_FrequencyFIC0

The *g_FrequencyFIC0* global variable contains the value of the FIC 0 clock frequency.

g_FrequencyFIC1

The *g_FrequencyFIC1* global variable contains the value of the FIC 1 clock frequency.

g_FrequencyFIC64

The *g_FrequencyFIC64* global variable contains the value of the FIC64 clock frequency.

Pre-boot code hardware configuration

The SmartFusion2 CMSIS-HAL provides a function prototype that can be used to implement hardware configuration that must be performed before the usual SmartFusion2 boot code is executed:

```
void mscc_post_hw_cfg_init(void)
```

This function is called after the DDR memory controller and SERDES are configured but before their reset line is de-asserted.

This function is intended to configure external clock sources used to clock the SERDES blocks. For example, it is used to configure the ZL30362 clock synthesizer on the SmartFusion2 Development Kit.

Redirecting standard library output to MMUART

The SmartFusion2 CMSIS-HAL allows redirecting the output of standard library functions such as printf to one of the MSS MMUART interfaces. This feature is enabled by adding one of the following defined symbols to your project settings:

MICROSEMI_STDIO_THRU_MMUART0

MICROSEMI_STDIO_THRU_MMUART1

Only one of these defined symbols should set in your project settings at any one time, depending on which MMUART you want to use.

A baud rate of 115200 is used by default for this feature. The baud rate can be modified using the MICROSEMI_STDIO_BAUD_RATE defined symbol. This defined symbol can be set to the desired baud rate in your project settings.

SoftConsole

Supported SoftConsole Versions

This version of the SmartFusion2 CMSIS requires SoftConsole v3.4 Service Pack 1 or a later version of SoftConsole.

Note: Note: GDB debug initialization scripts are only used with SoftConsole v3.4 and the section below detailing their use can be ignored if using SoftConsole v4.x or later.

Linker scripts

A number of linker scripts are provided with the SoftConsole SmartFusion2 CMSIS to cover some of the most common use cases. These scripts can be renamed and used as the starting point to create customized linker scripts for your applications.

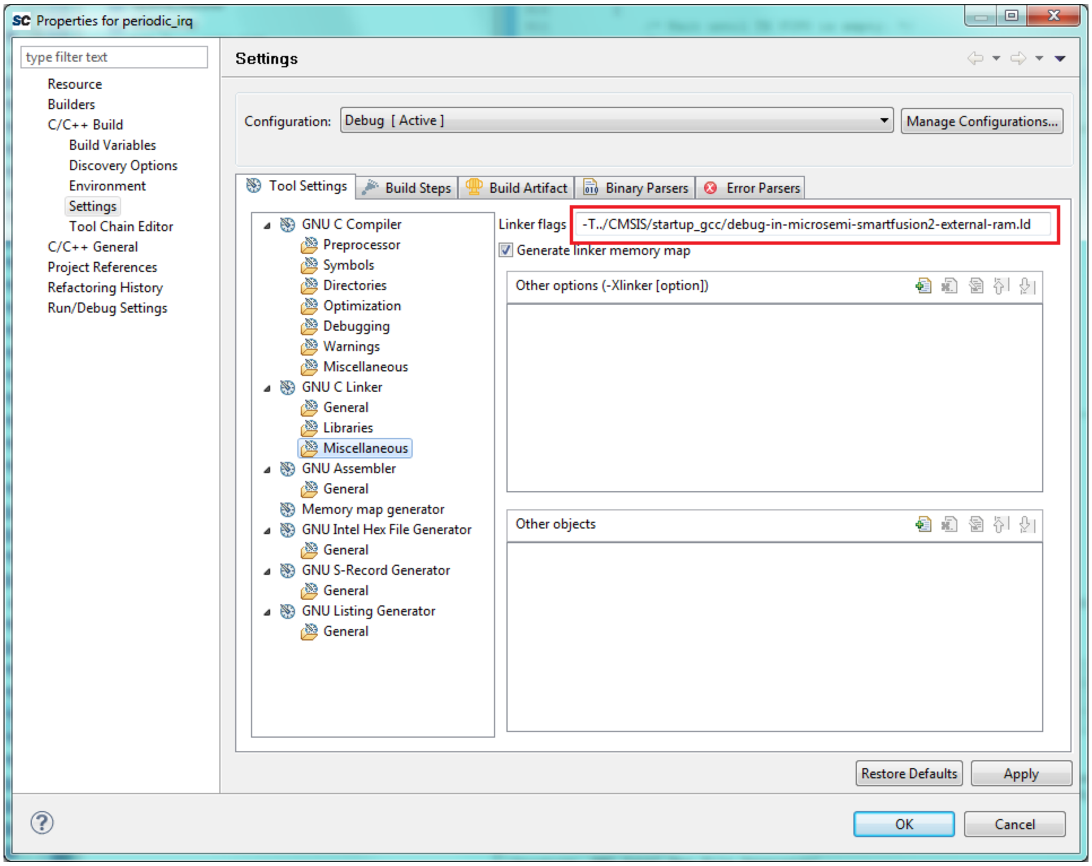
The SoftConsole linker scripts cover the following use cases:

- Debugging code executing in internal eSRAM
- Debugging code executing in internal eNVM
- Debugging code in external MDDR memory
- Production eNVM execute in place
- Production relocate executable from eNVM to MDDR

The debugging linker scripts are intended for use with the SoftConsole debugger. The production linker scripts are intended to be used to generate Intel-Hex files for use with the Libero hardware flow.

The SoftConsole linker scripts are located in the *CMSIS/startup_gcc* folder.

The linker script used to build your application is selected through the Project Properties as shown in the figure below.



Configuration: Debug [Active]

Linker flags: -T../CMSIS/startup_gcc/debug-in-microsemi-smartfusion2-external-ram.ld

☑ Generate linker memory map

Figure 2 · Selecting SoftConsole Linker Script

There are five linker scripts available:

- debug-in-microsemi-smartfusion2-esram.ld
- debug-in-microsemi-smartfusion2-envm.ld
- debug-in-microsemi-smartfusion2-external-ram.ld
- production-smartfusion2-execute-in-place.ld
- production-smartfusion2-relocate-to-external-ram.ld

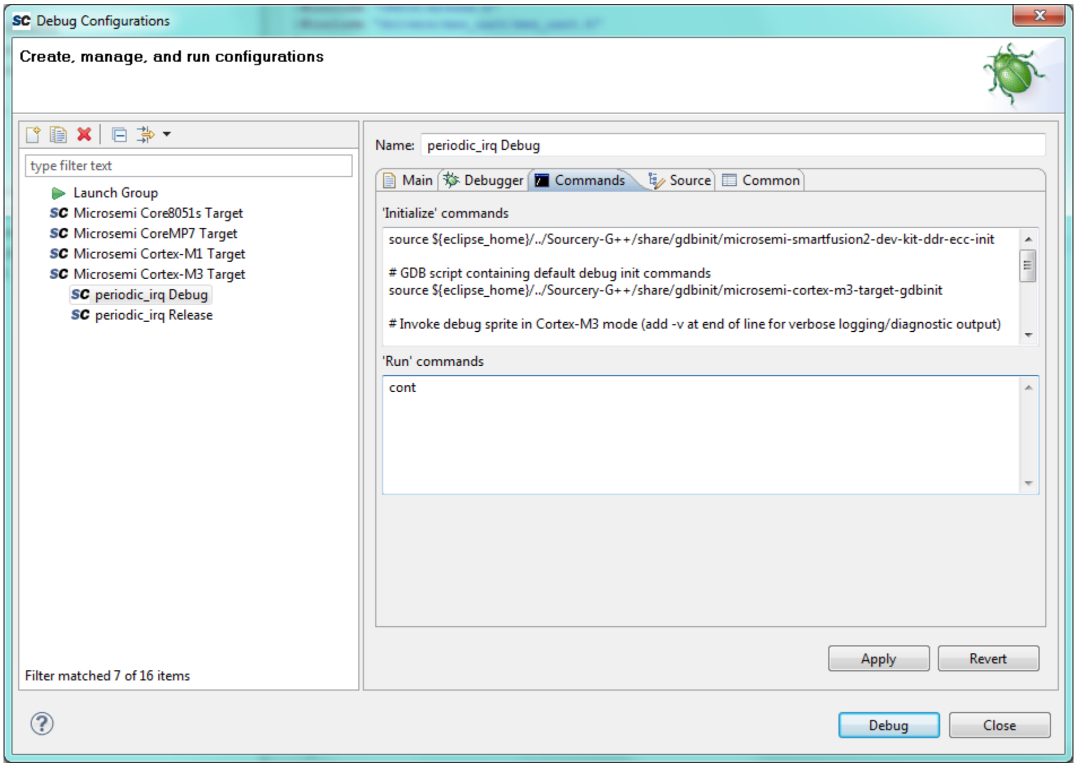
Debugging an executable running in external MDDR RAM

This version of the CMSIS-HAL allows building an executable for debugging in external MDDR RAM.

The application must be built using the *debug-in-microsemi-smartfusion2-external-ram.ld* linker script.

GDB debug initialization scripts (SoftConsole v3.4)

The debugger’s configuration needs to be customized to debug code running in external MDDR RAM in order to configure the MDDR controller before the debugger downloads the executable to the board.



Name: periodic_irq Debug

'Initialize' commands

```
source ${eclipse_home}/../Sourcery-G++/share/gdbinit/microsemi-smartfusion2-dev-kit-ddr-ecc-init

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

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

'Run' commands

```
cont
```

Figure 3 · SoftConsole debugger command for debugging code in MDDR

The commands are listed below.:

Line	Listing from example debug script
1	`source ${eclipse_home}/../Sourcery-G++/share/gdbinit/microsemi-smartfusion2-dev-kit-ddr-init`

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

3	`# Invoke debug sprite in Cortex-M3 mode (add -v at end of line for verbose logging/diagnostic output)`
	`target remote | "${eclipse_home}/../Sourcery-G++/bin/arm-none-eabi-sprite" flashpro:?cpu=Cortex-M3 "${build_loc}"`

4	`configure_external_ram`

5	`# Load the program (pre and post load hooks in gdbinit will be run)`
	`load`

The first line of the debugger commands specifies the GDB initialization script containing the configuration of the MDDR controller. This file is specific to the target hardware. Use the following table to choose the correct one.

DDR configuration script	When to use, and associated example linker files.
microsemi-smartfusion2-no-ddr-init	If not debugging from DDR use this. This will be the case when using the following linker files. debug-in-microsemi-smartfusion2-envm.ld debug-in-microsemi-smartfusion2-esram.ld

	production-smartfusion2-execute-in-place.ld production-smartfusion2-relocate-to-external-ram.ld
microsemi-smartfusion2-starter-kit-ddr-init	When using the starter kit. You will be using the following linker script. debug-in-microsemi-smartfusion2-external-ram.ld
microsemi-smartfusion2-eval-kit-ddr-init	When using the Eval kit. You will be using the following linker script. debug-in-microsemi-smartfusion2-external-ram.ld
microsemi-smartfusion2-dev-kit-ddr-init	When using the Development kit or advanced Development. kit. You will be using the following linker script. debug-in-microsemi-smartfusion2-external-ram.ld
microsemi-smartfusion2-dev-kit-ddr-ecc-init	When using the Development kit with ECC enabled. You will be using the following linker script. debug-in-microsemi-smartfusion2-external-ram.ld

Latest GDB debug initialization scripts (SoftConsole v3.4)

The GDB debug scripts referenced above come bundled with SoftConsole v3.4. They reside in the following directory:

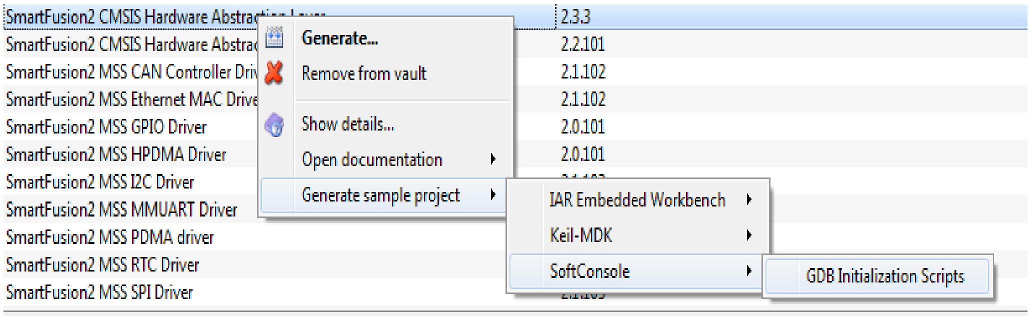
${eclipse_home}/../Sourcery-G++/share/gdbinit/

Note: *${eclipse_home}* is a macro containing the “root” SoftConsole v3.4 executable folder. For example, C:\Microsemi\Libero_v11.5\SoftConsole\ Eclipse.

The latest versions of these GDB scripts are included with the SmartFusion2 CMSIS- HAL.

They can be extracted using the sample project option under SmartFusion2 CMSIS-HAL in the Firmware Catalog and copied to *${eclipse_home}/../Sourcery-G++/share/gdbinit/*, overwriting the ones already there.

Note: When the files are extracted they are placed in a SoftConsole workspace subdirectory named *gdb_init_scripts*.



SmartFusion2 CMSIS Hardware Abstrac	2.3.3
SmartFusion2 CMSIS Hardware Abstrac	2.2.101
SmartFusion2 MSS CAN Controller Driv	2.1.102
SmartFusion2 MSS Ethernet MAC Drive	2.1.102
SmartFusion2 MSS GPIO Driver	2.0.101
SmartFusion2 MSS HPDMA Driver	2.0.101
SmartFusion2 MSS I2C Driver	
SmartFusion2 MSS MMUART Driver	
SmartFusion2 MSS PDMA driver	
SmartFusion2 MSS RTC Driver	
SmartFusion2 MSS SPI Driver	

Figure 4 · GDB scripts generation

When debugging, the following table gives which GDB debug initialization file should be used with which linker script given the hardware being used.

Boards with SmartFusion2 socket- **(1st May 2015)** **[http://www.microsemi.com/products/fpga-soc/design-resources/dev-kits/smartfusion2-kits]**				
Board	**SF2 device**	**MB**	**Associated GDB debug scripts included in CMSIS to be used when debugging external RAM.** **Note:** microsemi-cortex-m3-target-gdbinit is common to all.	**Associated linker script.** **These scripts assume the lowest common denominator amount of external RAM, 64MB.**
SF2-STARTER-KIT	M2S050-FGG484	64	microsemi-smartfusion2-starter-kit-ddr-init	debug-in-microsemi-smartfusion2-external-ram.ld
SF2-484-STARTER-KIT	M2S010-FGG484	64	microsemi-smartfusion2-starter-kit-ddr-init	debug-in-microsemi-smartfusion2-external-ram.ld
Security Eval. kit	M2S090TS-FGG484	64	microsemi-smartfusion2-eval-kit-ddr-init	debug-in-microsemi-smartfusion2-external-ram.ld
Advanced dev. kit	M2S150T-1FCG1152 ES	1GB	microsemi-smartfusion2-dev-kit-ddr-init or microsemi-smartfusion2-dev-kit-ddr-ecc-init	debug-in-microsemi-smartfusion2-external-ram.ld
Dual-Axis motor control kit	M2S010-FG484		n/a	n/a
dev. kit		512	microsemi-smartfusion2-dev-kit-ddr-init or microsemi-smartfusion2-dev-kit-ddr-ecc-init	debug-in-microsemi-smartfusion2-external-ram.ld
Going or gone obsolete				
Evaluation kit	M2S025T-1FGG484	64	microsemi-smartfusion2-eval-kit-ddr-init	debug-in-microsemi-smartfusion2-external-ram.ld

Table 3 · SmartFusion2 kits and related SoftConsole v3.4 scripts to use when debugging from DDR

IAR Embedded Workbench

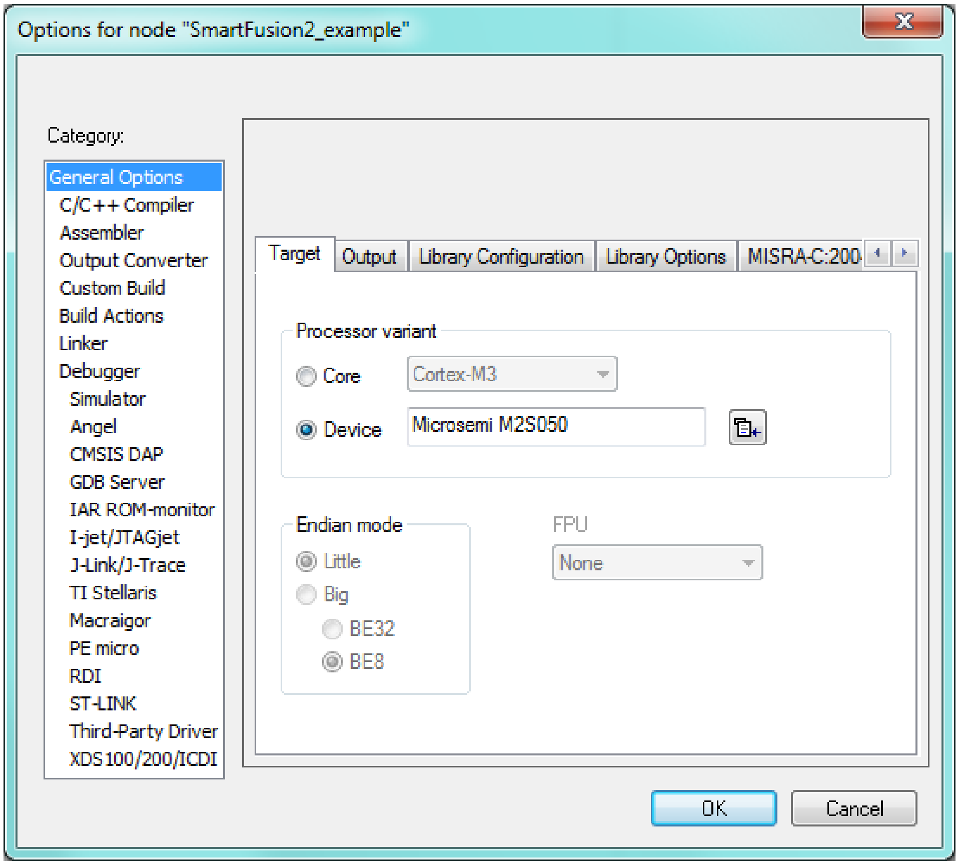
Creating a SmartFusion2 IAR Embedded Workbench Project

This section describes the steps required to create an IAR Embedded Workbench project targeted at a SmartFusion2 design.

SmartFusion2 devices are a highly configurable combination of a Cortex-M3 based microcontroller subsystem with a mainstream FPGA. The configuration complexity of SmartFusion2 is handled by the Libero SoC hardware design software tools. The difference between SmartFusion2 and a standard microcontroller is that we need to inject some of the Libero generated hardware configuration into the embedded software development flow. The merging of the hardware flow and embedded software development flow takes place in the CMSIS-HAL.

Select Target

Select one of the Microsemi SmartFusion2 devices as the target device in the project’s general options.



Options for node "SmartFusion2_example"

Processor variant

○ Core: Cortex-M3

● Device: Microsemi M2S050

Endian mode

● Little

○ Big

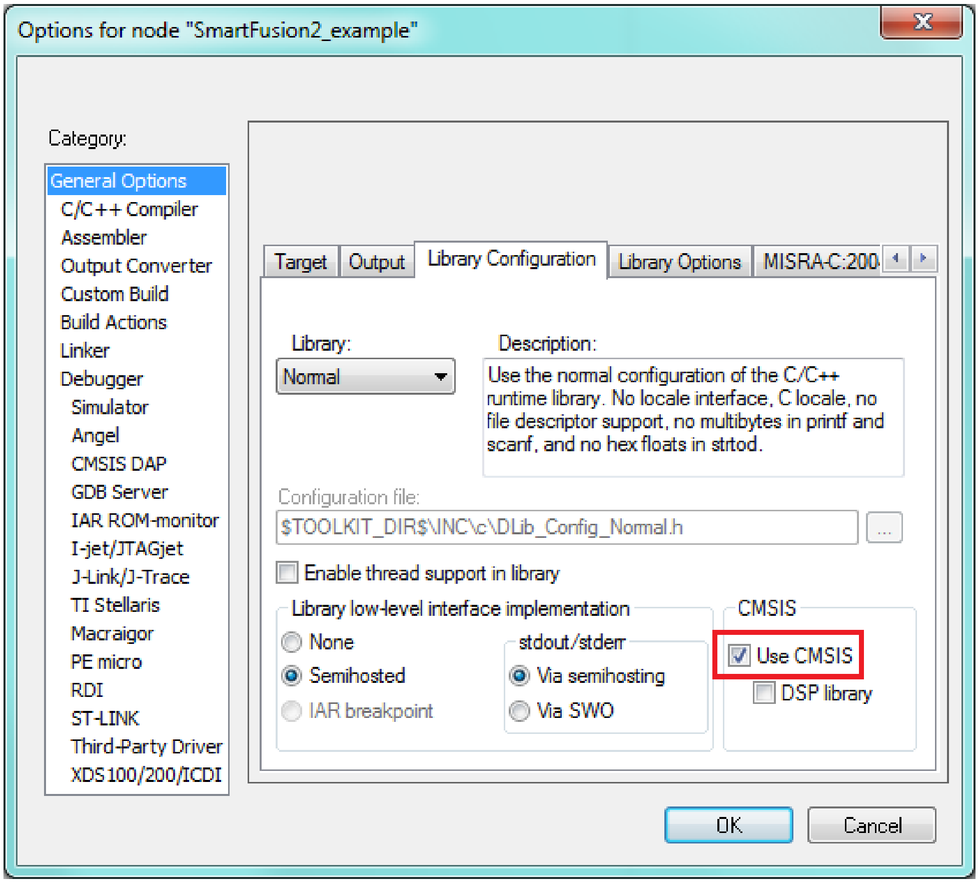
○ BE32

● BE8

FPU: None

Select “Use CMSIS” in Library Configuration

Select “Use CMSIS” in the project’s Library Configuration.



Options for node "SmartFusion2_example"

Category:

Library: Normal

Description: Use the normal configuration of the C/C++ runtime library. No locale interface, C locale, no file descriptor support, no multibytes in printf and scanf, and no hex floats in strtod.

Configuration file: $TOOLKIT_DIR$\INC\c\DLib_Config_Normal.h

□ Enable thread support in library

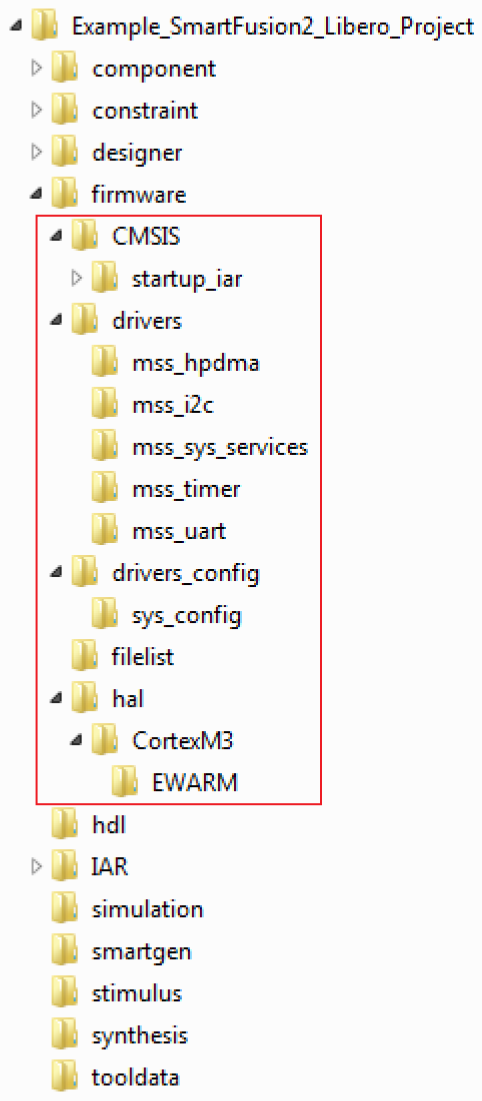
Library low-level interface implementation
○ None
● Semihosted
○ IAR breakpoint

stdout/stderr
● Via semihosting
○ Via SWO

CMSIS
☑ Use CMSIS
□ DSP library

Copy source code from Libero project

Copy the embedded software source code generated by the Libero project. This source code can be found in the firmware subfolder of the Libero project.



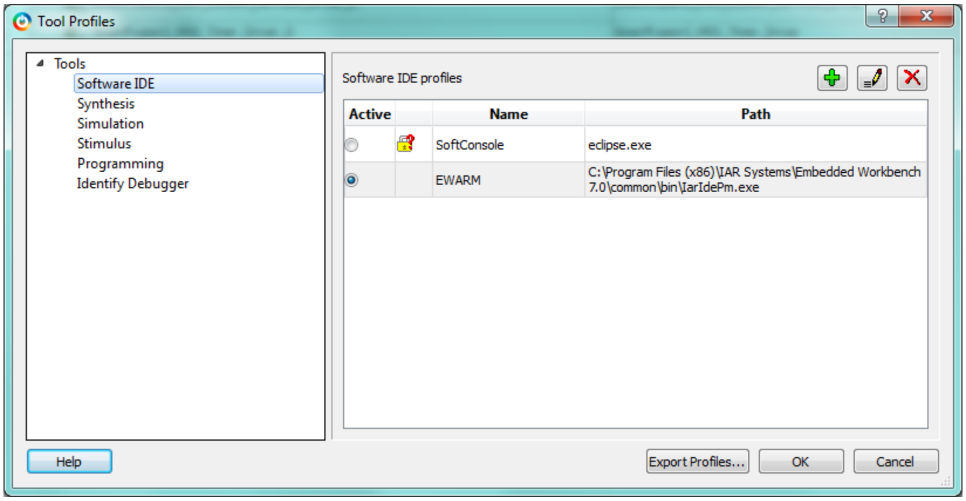
```
◢ Example_SmartFusion2_Libero_Project
    ▷ component
    ▷ constraint
    ▷ designer
    ◢ firmware
        ◢ CMSIS
            ▷ startup_iar
        ◢ drivers
              mss_hpdma
              mss_i2c
              mss_sys_services
              mss_timer
              mss_uart
        ◢ drivers_config
              sys_config
          filelist
        ◢ hal
            ◢ CortexM3
                  EWARM
      hdl
    ▷ IAR
      simulation
      smartgen
      stimulus
      synthesis
      tooldata
```

Figure 5 · Source files to copy from Libero project

Note: The *drivers/sys_config* folder contains configuration files generated by Libero specifically for your hardware project. It is critical that you keep the content of this folder synchronized with the Libero hardware design programmed on the target development board. Getting the content of this folder out of synch with the hardware design programmed on your development system may result in unexpected, difficult to debug, behaviour. Your first reflex on encountering unexpected system behaviour should be to ensure that your embedded software project uses the content of the *drivers_config/sys_config* folder of the Libero project used to program the SmartFusion2 device’s FPGA design.

Note: Please ensure that The IAR Embedded Workbench version of the SmartFusion2 CMSIS-HAL was generated by Libero. You can verify that the IAR Embedded Workbench version was generated by checking that the *firmware/CMSIS* folder contains a *startup_iar* subfolder and that the *firmware/hal/CortexM3* folder contains a *EWARM* subfolder.

You may need to modify the Libero tools profile to select IAR Embedded Workbench as the Software IDE in order for the IAR version of the CMSIS-HAL to be generated. This can be done using the Libero “Projects”->”Tool Profiles...” menu item.



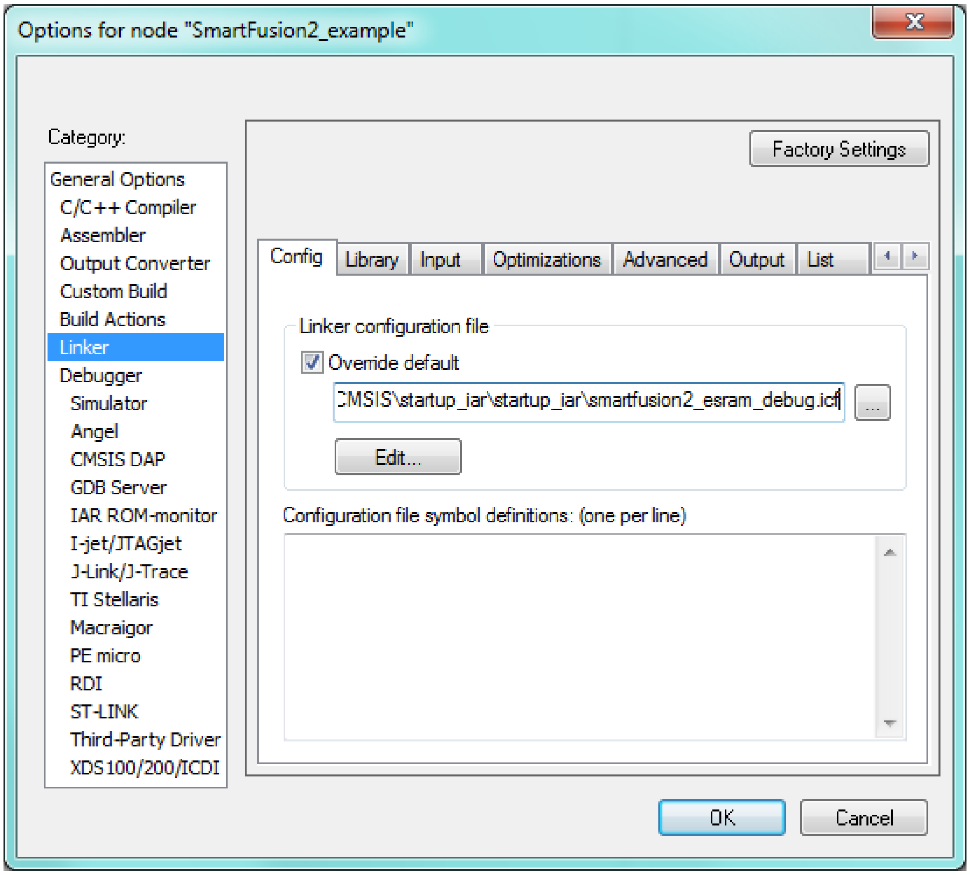
Active		Name	Path
○		SoftConsole	eclipse.exe
●		EWARM	C:\Program Files (x86)\IAR Systems\Embedded Workbench 7.0\common\bin\IarIdePm.exe

Figure 6 · Libero IAR Embedded Workbench Tools Profile

The embedded software source code can be configured and regenerated using the DESIGN_FIRMWARE Libero view of the design. This view is opened using the “Design”->”Configure Firmware” Libero menu item.

Set Linker Configuration File

A linker configuration file needs to be selected based on the type of target application. There are several options provided. Please refer to the “Linker Configuration Files” section.



Linker configuration file

☑ Override default

CMSIS\startup_iar\startup_iar\smartfusion2_esram_debug.icf

Configuration file symbol definitions: (one per line)

Please use $PROJ_DIR$ to specify the root project folder in the path to the linker configuration file to ensure that your project can be freely moved to different disk locations. For example, the eSRAM debug linker configuration file can be specified as follows:

```
$PROJ_DIR$\CMSIS\startup_iar\startup_iar\smartfusion2_esram_debug.icf
```

Migrating existing projects from CMSIS-HAL v2.1 or earlier

Some modifications are required to existing SmartFusion2 IAR Embedded Workbench projects in order to use the SmartFusion2 CMSIS-HAL v2.2.

The following new SmartFusion2 CMSIS-HAL files need to be added to the existing project:

- *vector_table.c*
- *default_irqs.c*
- *low_level_init.c*

The total list of SmartFusion2 CMSIS-HAL included in your project is:

- *startup_m2sxxx.s*
- *vector_table.c*
- *default_irqs.c*
- *low_level_init.c*
- *write.c*
- *read.c*

One of the linker configuration files provided with the SmartFusion2 CMSIS-HAL v2.2 firmware package must be used. The linker configuration file interacts with the start-up code. Not using one of the linker configuration files provided will result in some missing symbols required by the start-up code.

Linker Configuration Files

Several IAR Embedded Workbench linker configuration files are provided with the SmartFusion2 CMSIS-HAL:

- *smartfusion2_esram_debug.icf*
- *smartfusion2_envm.icf*
- *smartfusion2_mddr_debug.icf*
- *smartfusion2_envm_to_mddr.icf*

These configuration files cover some of the most common code and data location use cases. They can be used as is or copied and modified to suit your application's specific requirements.

smartfusion2_esram_debug.icf

The *smartfusion2_esram_debug.icf* linker configuration file is used to build a small debug application that will completely be executed from eSRAM.

smartfusion2_envm.icf

The *smartfusion2_envm.icf* linker configuration file is used to build applications executing from internal eNVM and only using internal eSRAM for read-write data. It allows building SmartFusion2 single chip applications.

smartfusion2_mddr_debug.icf

The *smartfusion2_mddr_debug.icf* linker configuration file is used to build large applications that will be executed from MDDR external memory. It allows debugging large applications executing in external MDDR memory. All executable code and data, including the heap, are located in MDDR memory. The internal eSRAM memory is only used by the C stack.

smartfusion2_envm_to_mddr.icf

The *smartfusion2_envm_to_mddr.icf* linker configuration file is used to build applications booting from eNVM and relocating to external MDDR memory for execution. All executable code and data, including the heap, are relocated to MDDR memory as part of the application start-up process. The internal eSRAM memory is only used by the C stack.

Note: You must also add the following defined symbol to your project settings for the vector table to be copied to DDR when relocating code from eNVM to DDR memory:
MSCC_RELOCATE_CODE_TO_EXT_RAM

Debugger Setup Macro Files

The SmartFusion2 CMSIS-HAL provides debugger setup macro files. These macro files are required for some use cases where the default target hardware configuration needs to be modified in order to successfully execute the debugged code. A debugger setup macro file is typically used to modify the Cortex-M3's vector table offset register to indicate the location of the vector table. It is also used to configure memory controllers prior to downloading the executable.

Three example debugger macro files are provided through the Firmware Catalog.

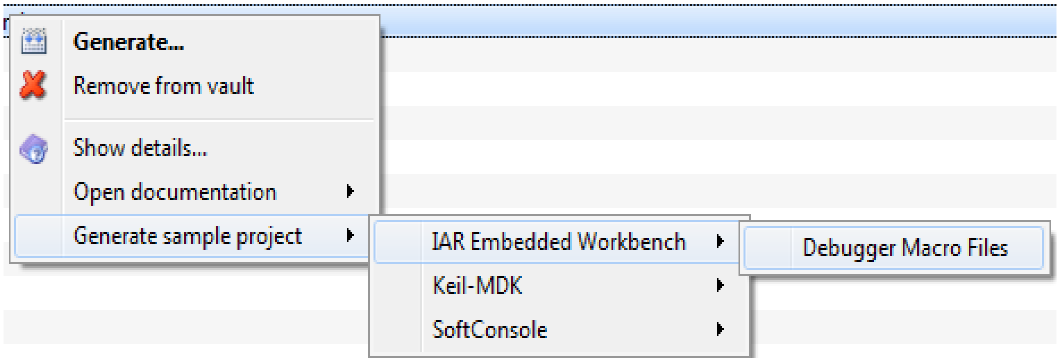
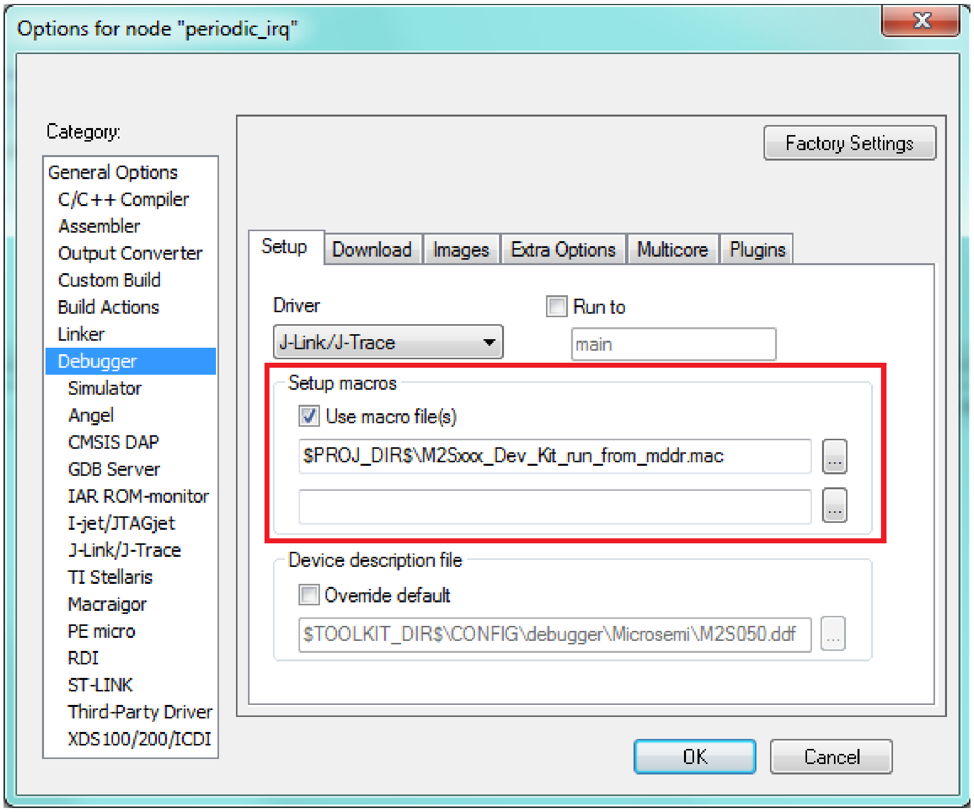


Figure 7 · Firmware Catalog Location of IAR Embedded Workbench Debugger Macro Files

The debugger setup macro is selected in the *Debugger* section of the project's settings as seen in the figure below:



Driver: J-Link/J-Trace

□ Run to: main

Setup macros

☑ Use macro file(s)

$PROJ_DIR$\M2Sxxx_Dev_Kit_run_from_mddr.mac

Device description file

□ Override default

$TOOLKIT_DIR$\CONFIG\debugger\Microsemi\M2S050.ddf

Figure 8 · Selecting Debugger Macro File

M2Sxxx_run_from_esram.mac

The *M2Sxxx_run_from_esram.mac* debugger setup macro file is intended to be used with applications built using the *smartfusion2_esram_debug.icf* linker configuration file. It allows debugging small applications completely fitting in eSRAM.

M2Sxxx_run_from_envm.mac

The *M2Sxxx_run_from_envm.mac* debugger setup macro file is intended to be used with applications built using the *smartfusion2_envm_debug.icf* linker configuration file. It allows debugging applications that use eNVM for code storage and eSRAM for read/write data.

M2Sxxx_Dev_Kit_run_from_mddr.mac

The *M2Sxxx_Dev_Kit_run_from_mddr.mac* debugger macro file is intended to be used with applications built using the *smartfusion2_mddr_debug.icf* linker configuration file. It allows debugging large applications executing in DDR memory connected to the MDDR memory controller. This debugger macro file configures the SmartFusion2 MDDR controller to use the DDR memory found on the SmartFusion2 Development Kit.

M2Sxxx_Starter_Kit_run_from_mddr.mac

The *M2Sxxx_Starter_Kit_run_from_mddr.mac* debugger macro file is intended to be used with applications built using the *smartfusion2_mddr_debug.icf* linker configuration file. It allows debugging large applications executing in DDR memory connected to the MDDR memory controller. This debugger macro file configures the SmartFusion2 MDDR controller to use the DDR memory found on the SmartFusion2 Starter Kit.

M2Sxxx_Eval_Kit_run_from_mddr.mac

The *M2Sxxx_Eval_Kit_run_from_mddr.mac* debugger macro file is intended to be used with applications built using the *smartfusion2_mddr_debug.icf* linker configuration file. It allows debugging large applications executing in DDR memory connected to the MDDR memory controller. This debugger macro file configures the SmartFusion2 MDDR controller to use the DDR memory found on the SmartFusion2 Eval Kit.

When debugging, the following table gives which IAR debug file should be used with which linker script given the hardware being used.

Boards with SmartFusion2 socket- (1st May 2015) [http://www.microsemi.com/products/fpga-soc/design-resources/dev-kits/smartfusion2-kits]				
Board	**SF2 device**	**MB**	**Associated IAR debug scripts included in CMSIS to be used when debugging external RAM.**	**Associated IAR linker script** **These scripts assume the lowest common denominator amount of external RAM, 64MB.**
SF2-STARTER-KIT	M2S050-FGG484	64	M2Sxxx_Starter_Kit_run_from_mddr.mac	smartfusion2_mddr_debug.icf
SF2-484-STARTER-KIT	M2S010-FGG484	64	M2Sxxx_Starter_Kit_run_from_mddr.mac	debug-in-microsemi-smartfusion2-external-ram.ld
Security Eval. kit	M2S090TS-FGG484	64	M2Sxxx_Eval_Kit_run_from_mddr.mac	smartfusion2_mddr_debug.icf
Advanced dev. kit	M2S150T-1FCG1152ES	1GB	M2Sxxx_Dev_Kit_run_from_mddr.mac	smartfusion2_mddr_debug.icf

Dual-Axis motor control kit	M2S010-FG484		n/a	n/a
dev. kit	M2S050T-1FGG896	512	M2Sxxx_Dev_Kit_run_from_mddr.mac	smartfusion2_mddr_debug.icf
Going or gone obsolete				
Evaluation kit	M2S025T-1FGG484	64	M2Sxxx_Eval_Kit_run_from_mddr.mac	smartfusion2_mddr_debug.icf

Table 4 · SmartFusion2 kits and related IAR files to use when debugging from DDR

M2Sxxx.svd

This file contains detailed descriptions of all possible registers used in the SmartFusion2 MSS. It is in a format defined by ARM and can be imported into IAR to allow viewing and setting of all possible registers when debugging.

Keil-MDK

Legacy support

To use SmartFusion2 projects with latest version of Keil-MDK (v5.14), legacy support must be installed. This can be downloaded from the Keil website.

Linker Scatter Files

Several Keil-MDK linker scatter files are provided with the SmartFusion2 CMSIS-HAL:

- smartfusion2_esram_debug.sct
- smartfusion2_execute_in_place.sct
- smartfusion2_mddr_debug.sct
- smartfusion2_relocate_to_external_ram.sct

These linker scatter files cover some of the most common code and data location use cases. They can be used as is or copied and modified to suit your application's specific requirements.

smartfusion2_esram_debug.sct

The *smartfusion2_esram_debug.sct* linker scatter file is used to build a small debug application that will completely be executed from eSRAM. The eSRAM is divided equally between code and read write values.

smartfusion2_execute_in_place.sct

The *smartfusion2_execute_in_place.sct* linker scatter file is used to build applications executing from internal eNVM and only using internal eSRAM for read-write data. It allows building SmartFusion2 single chip applications.

smartfusion2_mddr_debug.sct

The *smartfusion2_mddr_debug.sct* linker scatter file is used to build large applications that will be executed from MDDR external memory. It allows debugging large applications executing in external MDDR memory. All executable code and data, including the heap, are located in MDDR memory. The internal eSRAM memory is only used by the C stack.

smartfusion2_relocate_to_external_ram.sct

The *smartfusion2_relocate to_external_ram.sct* linker scatter file is used to build applications booting from eNVM and relocating to external MDDR memory for execution. All executable code and data, including the heap, are relocated to MDDR memory as part of the application start-up process. The internal eSRAM memory is only used by the C stack.

Debugger Initialization Files

The SmartFusion2 CMSIS-HAL provides Keil-MDK debugger initialization files. These debugger initialization files are required for some use cases where the default target hardware configuration needs to be modified in order to successfully execute the debugged code. A debugger initialization file is typically used to modify the Cortex-M3's vector table offset register to indicate the location of the vector table. It is also used to configure memory controllers prior to downloading the executable.

Five Keil-MDK debugger initialization files are provided:

- M2Sxxx_esram.ini
- M2Sxxx_envm.ini
- M2Sxxx_mddr_dev_kit.ini
- M2Sxxx_mddr_eval_kit.ini
- M2Sxxx_mddr_starter_kit.ini

Similar to IAR, these files are obtained using the sample project option as shown below:

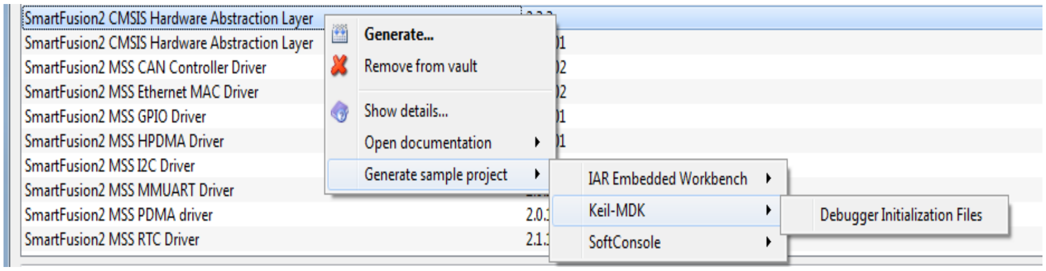
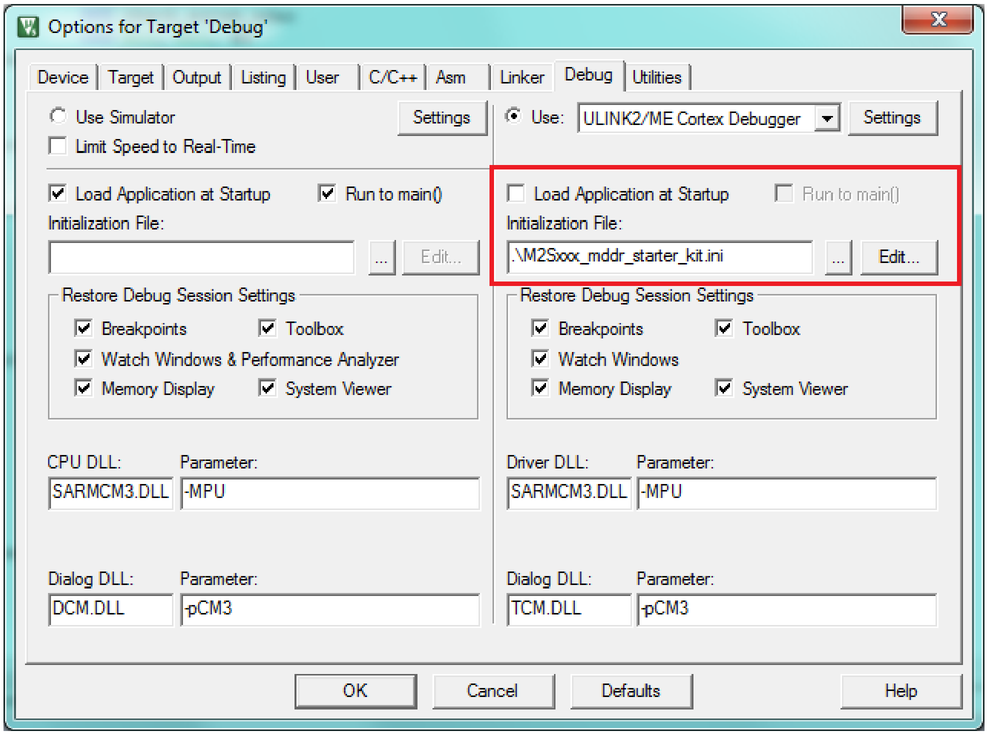

SmartFusion2 CMSIS Hardware Abstraction Layer
SmartFusion2 CMSIS Hardware Abstraction Layer
SmartFusion2 MSS CAN Controller Driver
SmartFusion2 MSS Ethernet MAC Driver
SmartFusion2 MSS GPIO Driver
SmartFusion2 MSS HPDMA Driver
SmartFusion2 MSS I2C Driver
SmartFusion2 MSS MMUART Driver
SmartFusion2 MSS PDMA driver
SmartFusion2 MSS RTC Driver
Generate...
Remove from vault
Show details...
Open documentation
Generate sample project
IAR Embedded Workbench
Keil-MDK
SoftConsole
Debugger Initialization Files

Figure 9 · KEIL debugger scripts generation

Keil-MDK debugger initializations files are selected in the Debug tab of the project's settings as seen in the figure below.



○ Use Simulator
□ Limit Speed to Real-Time
☑ Load Application at Startup ☑ Run to main()
Initialization File:
☑ Breakpoints ☑ Toolbox
☑ Watch Windows & Performance Analyzer
☑ Memory Display ☑ System Viewer
CPU DLL: SARMCM3.DLL Parameter: -MPU
Dialog DLL: DCM.DLL Parameter: -pCM3

● Use: ULINK2/ME Cortex Debugger
□ Load Application at Startup □ Run to main()
Initialization File: .\M2Sxxx_mddr_starter_kit.ini
☑ Breakpoints ☑ Toolbox
☑ Watch Windows
☑ Memory Display ☑ System Viewer
Driver DLL: SARMCM3.DLL Parameter: -MPU
Dialog DLL: TCM.DLL Parameter: -pCM3

Figure 10 · Keil-MDK Debugger Initialization File Setting

Note: Some of the debugger initialization files control the loading of the executable image to the target hardware. It is therefore sometime necessary to un-tick the *Load Application at Startup* tick-box.

If the debugger file contains the line below, untick the *Load Application at Startup* tick-box.

```
LOAD %L INCREMENTAL
```

M2Sxxx_esram.ini

The *M2Sxxx_esram.ini* debugger initialization file is intended to be used with applications built using the *smartfusion2_esram_debug.sct* linker scatter file. It allows debugging small applications completely fitting in eSRAM.

This debugger initialization file does not handle the loading of the executable to be debugged to the target hardware. It is therefore necessary to make sure that *Load Application at Startup* is ticked when using this file.

M2Sxxx_envm.ini

The *M2Sxxx_envm.ini* debugger initialization file is intended to be used with applications built using the *smartfusion2_envm_debug.sct* linker scatter file. It allows debugging application running from eNVM including relocation applications that transfer to DDR after bootup.

This debugger initialization file does not handle the loading of the executable to be debugged to the target hardware. It is therefore necessary to make sure that *Load Application at Startup* is ticked when using this file.

M2Sxxx_mddr_dev_kit.ini

The *M2Sxxx_Dev_Kit_run_from_mddr.mac* debugger initialization file is intended to be used with applications built using the *smartfusion2_mddr_debug.sct* linker scatter file. It allows debugging large applications executing in DDR memory connected to the MDDR memory controller. This debugger macro file configures the SmartFusion2 MDDR controller to use the DDR memory found on the SmartFusion2 Development Kit.

This debugger initialization file handles the loading of the executable to be debugged to the target hardware. It is therefore necessary to make sure that *Load Application at Startup* is not ticked when using this file.

M2Sxxx_mddr_starter_kit.ini

The *M2Sxxx_Starter_Kit_run_from_mddr.mac* debugger initialization file is intended to be used with applications built using the *smartfusion2_mddr_debug.sct* linker scatter file. It allows debugging large applications executing in DDR memory connected to the MDDR memory controller. This debugger macro file configures the SmartFusion2 MDDR controller to use the DDR memory found on the SmartFusion2 Starter Kit.

This debugger initialization file handles the loading of the executable to be debugged to the target hardware. It is therefore necessary to make sure that *Load Application at Startup* is not ticked when using this file.

M2Sxxx_mddr_eval_kit.ini

The *M2Sxxx_Starter_Kit_run_from_mddr.mac* debugger initialization file is intended to be used with applications built using the *smartfusion2_mddr_debug.sct* linker scatter file. It allows debugging large applications executing in DDR memory connected to the MDDR memory controller. This debugger macro file configures the SmartFusion2 MDDR controller to use the DDR memory found on the SmartFusion2 Eval Kits.

This debugger initialization file handles the loading of the executable to be debugged to the target hardware. It is therefore necessary to make sure that *Load Application at Startup* is not ticked when using this file.

When debugging, the following table gives which debug file should be used when debugging from external memory, given the hardware being used.

Boards with SmartFusion2 socket- 1st May 2015 [http://www.microsemi.com/products/fpga-soc/design-resources/dev-kits/smartfusion2-kits]				
Board	**SF2 device**	**MB**	**Associated KEIL debug scripts**	**Associated KEIL Linker script** **This script should assume LCD amount of external RAM is 64MB**
SF2-STARTER-KIT	M2S050-FGG484	64	M2Sxxx_mddr_starter_kit.ini	smartfusion2_mddr_debug.sct
SF2-484-STARTER-KIT	M2S010-FGG484	64	M2Sxxx_mddr_starter_kit.ini	smartfusion2_mddr_debug.sct
Security Eval. kit	M2S090TS-FGG484	64	M2Sxxx_mddr_eval_kit.ini	smartfusion2_mddr_debug.sct
Advanced dev. kit	M2S150T-1FCG1152ES	1GB	M2Sxxx_mddr_dev_kit.ini	smartfusion2_mddr_debug.sct
Dual-Axis motor control kit	M2S010-FG484		n/a	n/a
dev. kit	M2S050T-1FGG896	512	M2Sxxx_mddr_dev_kit.ini	smartfusion2_mddr_debug.sct
Going or gone obsolete				
Evaluation kit	M2S025T-1FGG484	64	M2Sxxx_mddr_eval_kit.ini	smartfusion2_mddr_debug.sct

Table 5 · SmartFusion2 kits and related IAR files to use when debugging from DDR

M2Sxxx.SFR

This file contains detailed descriptions of all possible registers used in the SmartFusion2 MSS. It is in a format defined by Keil and can be imported into Keil-MDK to allow viewing and setting of all possible registers when debugging.

Problem Solving

Watchdog interrupt occurs when debugging

The watchdog can be enabled in hardware when creating a Libero project. When debugging, the watchdog must be disabled. The disabling of the watchdog has been added to all the example debug scripts for SoftConsole, Keil and IAR.

The watchdog can also be enabled by firmware. This watchdog enabling code must not be included in debug mode.

An example of how to exclude this code when compiling for Debug is shown below:

```
#if defined(NDEBUG)
    SYSREG->WDOG_CR = 0;
#else
    /* turn on watchdog here if required */
#endif
```

Unexpected values seen in memory when debugging

The cache can be enabled in hardware when creating a Libero project. When debugging, the cache should be disabled. The disabling of the cache has been added to all the example debug scripts for SoftConsole, Keil and IAR.

The cache can also be enabled by firmware. This cache enabling code must not be included in debug mode.

An example of how to exclude this code when compiling for Debug is shown below:

```
#if defined(NDEBUG)
    SYSREG->CC_CR = 1u;
#else
    SYSREG->CC_CR = 0u;
#endif
```

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

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at **www.microsemi.com**.