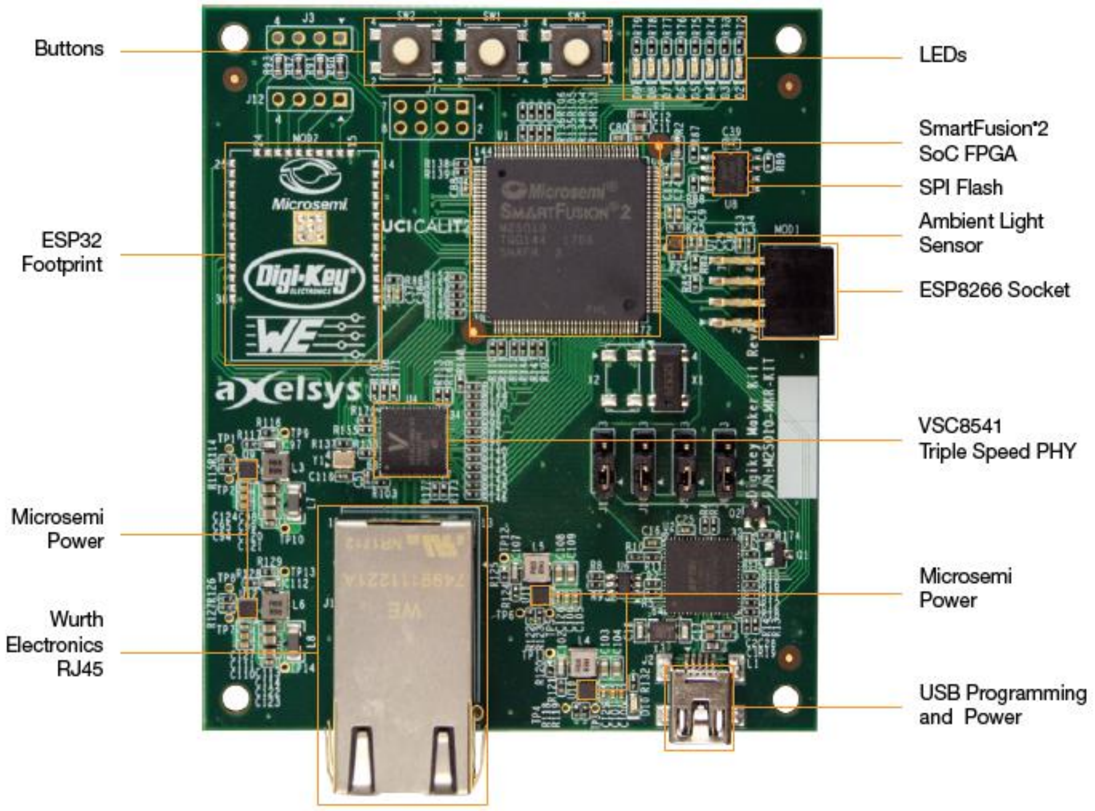
SMARTFUSION2 MAKER GUIDE

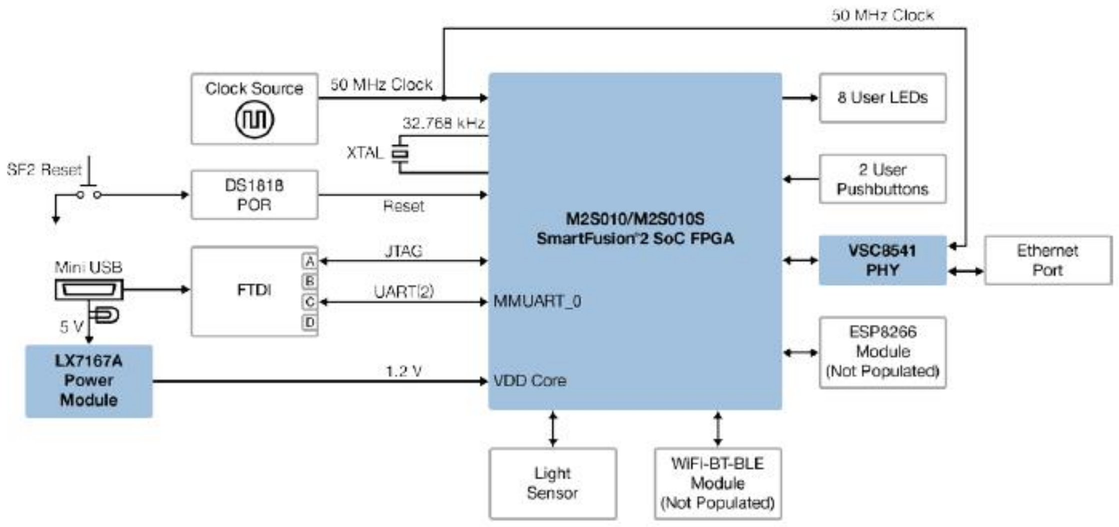
2021/02/28 REVISION 1.0

TABLE OF CONTENTS

INTRODUCTION

This lab exercise gives you the opportunity to bring up and program both the hardware and software of the Microsemi SmartFusion2 System on a Chip (SoC). To do this we will use the Digi-key SmartFusion2 Maker Kit shown below. It includes the SmartFusion2 which has 12K LE flash-based FPGA fabric, a 166 MHz ARM Cortex-M3 processor, DSP blocks, SRAM, eNVM, and general purpose GPIO interfaces. The board includes many interfaces including a USB port for JTAG programming, UART communications, and powering the board; a SPI flash; a 50 MHz clock source; 8 user LEDS; 2 user pushbuttons; a light sensor; and the VSC8541 PHY for 100 Mbps/1 Gbps Ethernet. There are also two unpopulated laid out connections—these pinouts support an ESP8266 and an ESP32 Wi-Fi/Bluetooth module (not included). You will exercise some of these interfaces in this exercise.





LEARNING OBJECTIVES

For this project, the objective is for students to:

- Become familiar with the Microsemi FPGA development flow and use of the Libero
- Appreciate the capabilities of the Microsemi SmartFusion2 System on a chip
- Learn how to create hardware using the SmartDesign design tool.
- Learn how to develop Software using the SoftConsole IDE
- Learn to program the SmartFusion2 Maker Board over JTAG.
- Practice debugging hardware and software together

RESOURCES

Resources for this exercise that may be useful to you can be found on Canvas including:

- M2S010-MKR-KIT Schematics The schematics for the M2S010-MKR-KIT board can be downloaded from Digikey
- Software development files in SF2Maker.zip
- SoftConsole Release Notes
- SmartFusion2 datasheet
- SmartFusion2 Maker Kit Quick Start Guide

I. VERIFY SMARTFUSION2 MAKER BOARD

Apply power to the SmartFusion2 Maker board by connecting it to a PC using a mini-USB cable. When the Digikey SmartFusion2 Maker Board is powered on, all 8 LEDs will flash on and off repeatedly. Once user button 1 is pressed, the ambient light sensor will be continuously read. The result of the light sensor reading will be shown visually across the 8 LEDs as a bar graph. Confirm that the board behaves as expected.

II. CREATE A HARDWARE DESIGN WITH LIBERO

SETUP

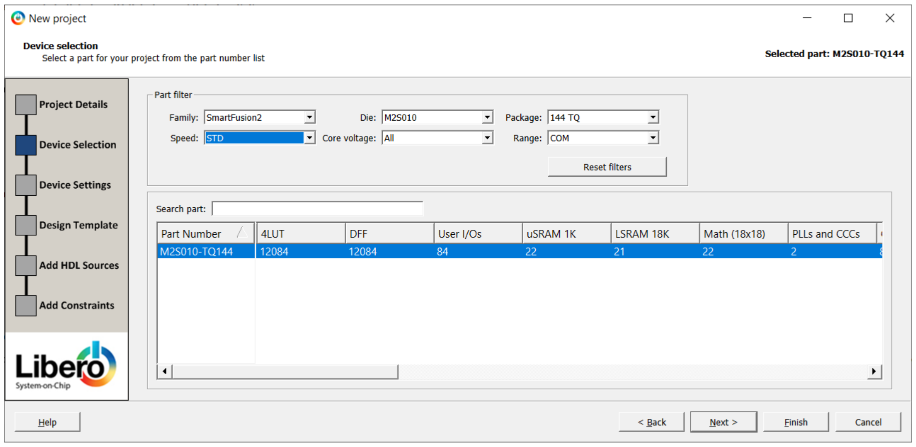
To set up your project:

1. Invoke Libero SoC. From the **Project** menu, choose **New Project**.

2. Enter SF2MakerDemo for your new project name and browse to a folder for your project location. Enter the following values for the project settings:

- Language: Verilog
- Family: SmartFusion2
- Package: 144 TQFP
- Speed: STD

- Die: M2S010
- Temperature Range: COM
- I/O: LVCMOS 2.5v
- PLL Supply Voltage: 2.5
- VDD supply ramp time: 100 ms.
- Design Templates: System Builder based Design
- HDL Source File: TBD from the file provided



New project

Device selection
Select a part for your project from the part number list

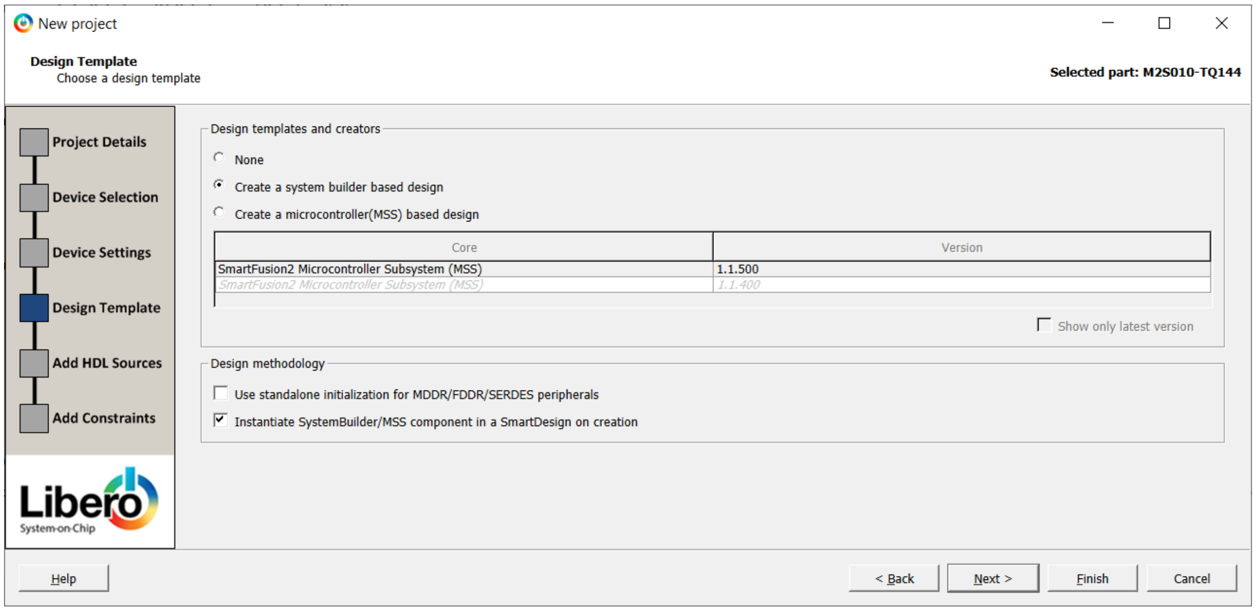
Selected part: M2S010-TQ144

Part filter

Family: SmartFusion2
Die: M2S010
Package: 144 TQ
Speed: STD
Core voltage: All
Range: COM

Search part:

Part Number	4LUT	DFF	User I/Os	uSRAM 1K	LSRAM 18K	Math (18x18)	PLLs and CCCs
M2S010-TQ144	12084	12084	84	22	21	22	2



New project

Design Template
Choose a design template

Selected part: M2S010-TQ144

Design templates and creators

None
[✓ Create a system builder based design]
Create a microcontroller(MSS) based design

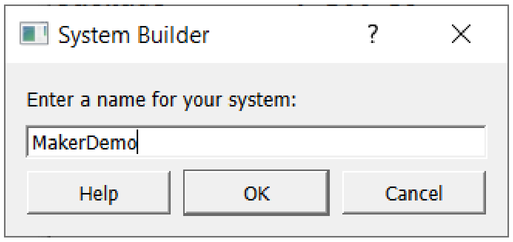
Core	Version
SmartFusion2 Microcontroller Subsystem (MSS)	1.1.500
SmartFusion2 Microcontroller Subsystem (MSS)	1.1.400

Show only latest version

Design methodology

Use standalone initialization for MDDR/FDDR/SERDES peripherals
[✓ Instantiate SystemBuilder/MSS component in a SmartDesign on creation]

3. No HDL sources or constraints will be added at this time. Click **Finish**.
4. If you are prompted to select a constraint flow, choose **Enhanced Constraint Flow**.
5. Libero will ask you to name the system. Give it the name MakerDemo and click **OK**.

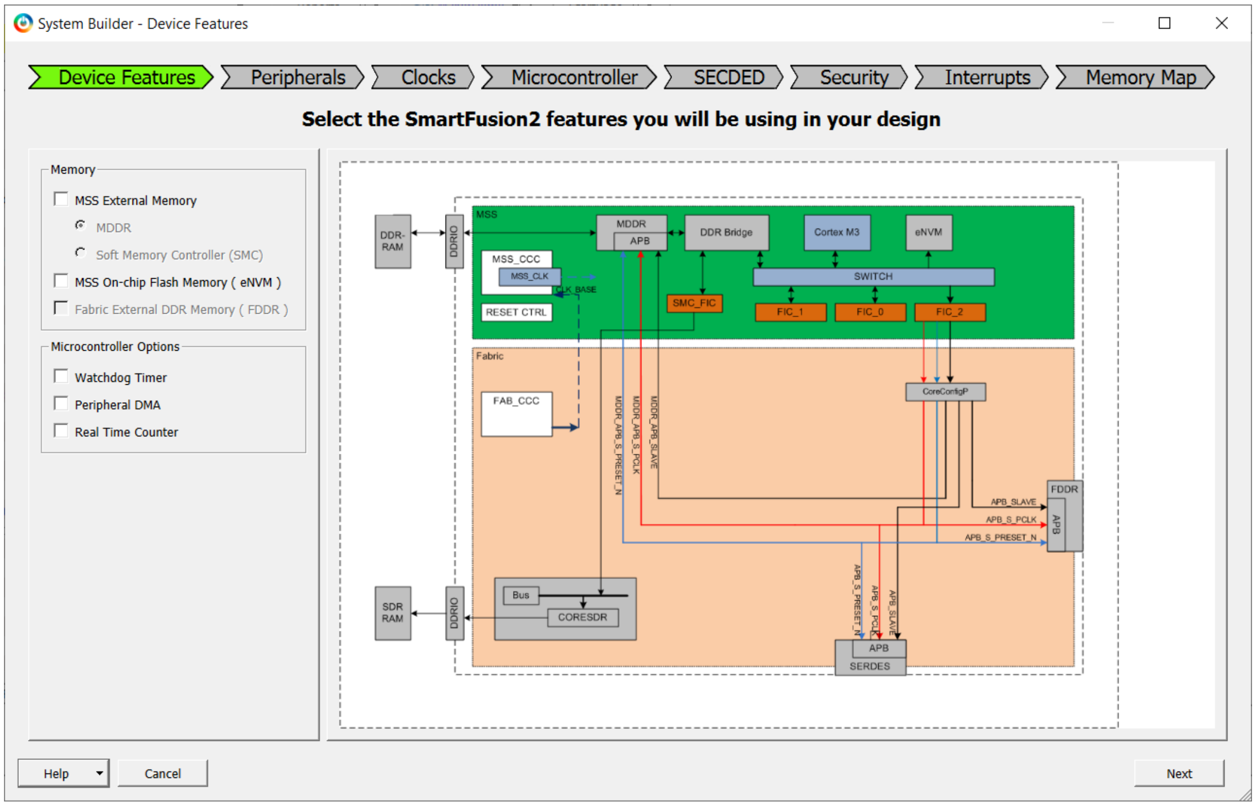

System Builder

Enter a name for your system:

MakerDemo

BUILD THE SYSTEM

6. The System Builder startup window will appear. It allows you to set up the SmartFusion2 by configuring all the devices in the MSS and the Fabric. Click **Next** to accept the defaults.

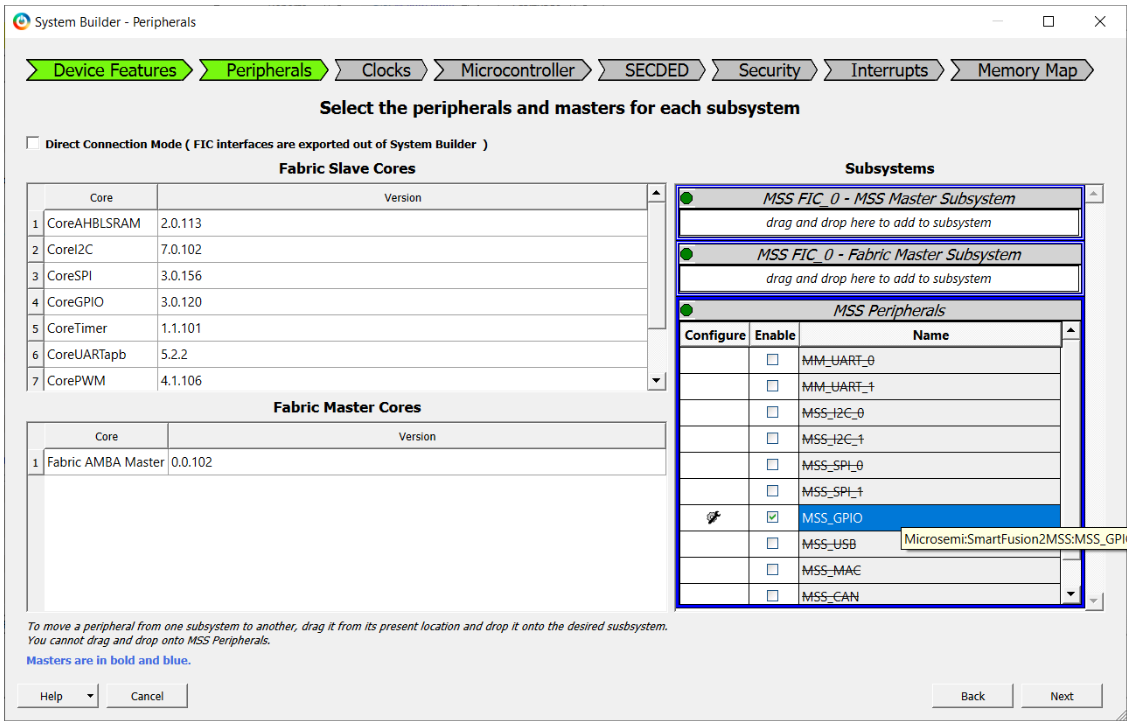

System Builder - Device Features

Select the SmartFusion2 features you will be using in your design

Memory
- ☐ MSS External Memory
  - MDDR
  - Soft Memory Controller (SMC)
- ☐ MSS On-chip Flash Memory (eNVM)
- ☐ Fabric External DDR Memory (FDDR)

Microcontroller Options
- ☐ Watchdog Timer
- ☐ Peripheral DMA
- ☐ Real Time Counter

7. In the **Peripherals** sub-window under System Builder, in the **Subsystems** pane, disable all peripherals. Enable **MSS_GPIO** and click the configuration button next to the **Enable** tick-box.



System Builder - Peripherals

Device Features | Peripherals | Clocks | Microcontroller | SECDED | Security | Interrupts | Memory Map

Select the peripherals and masters for each subsystem

☐ Direct Connection Mode (FIC interfaces are exported out of System Builder)

Fabric Slave Cores

	Core	Version
1	CoreAHBLSRAM	2.0.113
2	CoreI2C	7.0.102
3	CoreSPI	3.0.156
4	CoreGPIO	3.0.120
5	CoreTimer	1.1.101
6	CoreUARTapb	5.2.2
7	CorePWM	4.1.106

Fabric Master Cores

	Core	Version
1	Fabric AMBA Master	0.0.102

Subsystems

MSS FIC_0 - MSS Master Subsystem
drag and drop here to add to subsystem

MSS FIC_0 - Fabric Master Subsystem
drag and drop here to add to subsystem

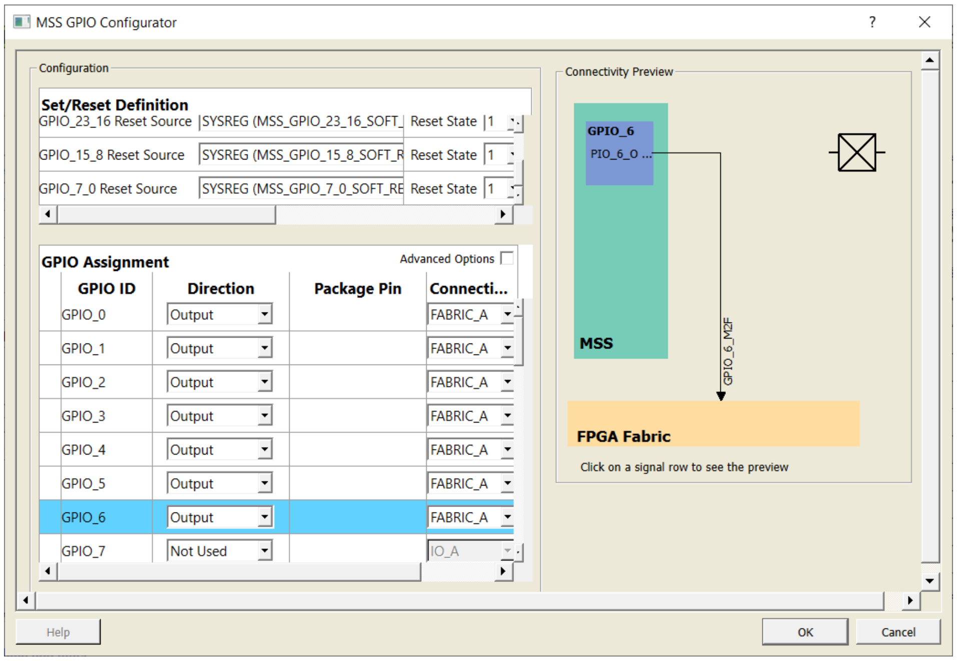
MSS Peripherals

Configure	Enable	Name
	☐	~~MM_UART_0~~
	☐	~~MM_UART_1~~
	☐	~~MSS_I2C_0~~
	☐	~~MSS_I2C_1~~
	☐	~~MSS_SPI_0~~
	☐	~~MSS_SPI_1~~
🔧	✓	MSS_GPIO
	☐	~~MSS_USB~~
	☐	~~MSS_MAC~~
	☐	~~MSS_CAN~~

Microsemi:SmartFusion2MSS:MSS_GPI

To move a peripheral from one subsystem to another, drag it from its present location and drop it onto the desired subsystem.
You cannot drag and drop onto MSS Peripherals.
Masters are in bold and blue.

8. Click on the wrench in the Configure Column next to the MSS_GPIO. Configure GPIO_0 through GPIO_7 as outputs connected to the fabric, as shown below:



MSS GPIO Configurator

Configuration

Set/Reset Definition

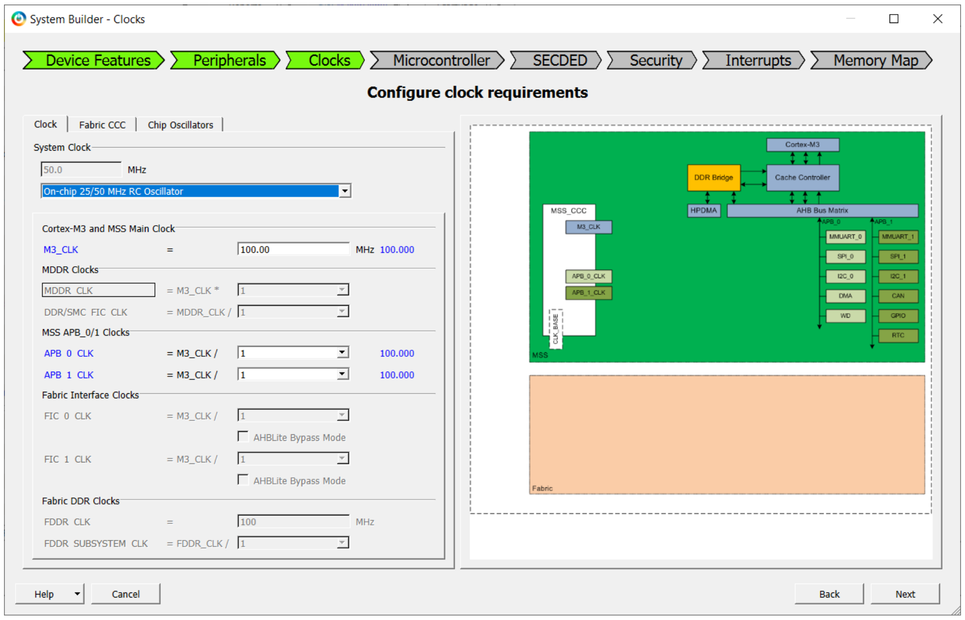
GPIO_23_16 Reset Source	SYSREG (MSS_GPIO_23_16_SOFT_	Reset State	1
GPIO_15_8 Reset Source	SYSREG (MSS_GPIO_15_8_SOFT_R	Reset State	1
GPIO_7_0 Reset Source	SYSREG (MSS_GPIO_7_0_SOFT_RE	Reset State	1

GPIO Assignment Advanced Options ☐

GPIO ID	Direction	Package Pin	Connecti...
GPIO_0	Output		FABRIC_A
GPIO_1	Output		FABRIC_A
GPIO_2	Output		FABRIC_A
GPIO_3	Output		FABRIC_A
GPIO_4	Output		FABRIC_A
GPIO_5	Output		FABRIC_A
GPIO_6	Output		FABRIC_A
GPIO_7	Not Used		IO_A

9. Click **OK** in the MSS GPIO Configurator window and click **Next** back in System Builder.

10. In the Clocks sub-window in System Builder, select **On-chip 25/50 MHz RC Oscillator**. Leave the rest of the settings to default (M3_CLK = 100 MHz) and click **Next**.



System Builder - Clocks

Device Features > Peripherals > Clocks > Microcontroller > SECDED > Security > Interrupts > Memory Map

Configure clock requirements

Clock | Fabric CCC | Chip Oscillators

System Clock
50.0 MHz
On-chip 25/50 MHz RC Oscillator

Cortex-M3 and MSS Main Clock
M3_CLK = 100.00 MHz 100.000

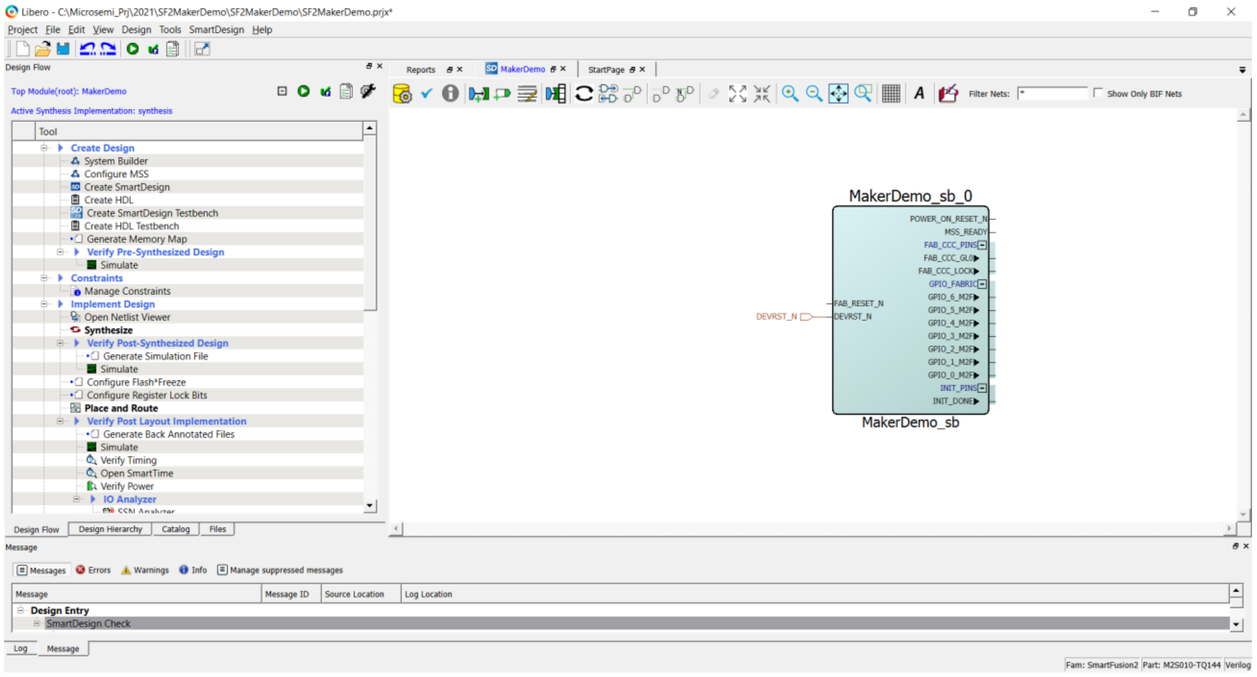
MDDR Clocks
MDDR_CLK = M3_CLK * 1
DDR/SMC_FIC_CLK = MDDR_CLK / 1

MSS APB_0/1 Clocks
APB_0_CLK = M3_CLK / 1 100.000
APB_1_CLK = M3_CLK / 1 100.000

Fabric Interface Clocks
FIC_0_CLK = M3_CLK / 1
AHBLite Bypass Mode
FIC_1_CLK = M3_CLK / 1
AHBLite Bypass Mode

Fabric DDR Clocks
FDDR_CLK = 100 MHz
FDDR_SUBSYSTEM_CLK = FDDR_CLK / 1

11. Click **Next** in the Microcontroller sub-window to accept the defaults.
12. Click **Next** in the SECDED sub-window to accept the defaults.
13. Click **Next** in the Security sub-window to accept the defaults.
14. Click **Next** in the Interrupts sub-window to accept the defaults.
15. Click **Finish** in the Memory Map window. The System Builder design should now be finished.
16. The generated MSS system should now appear in SmartDesign.



Libero - C:\Microsemi_Prj\2021\SF2MakerDemo\SF2MakerDemo\SF2MakerDemo.prjx*

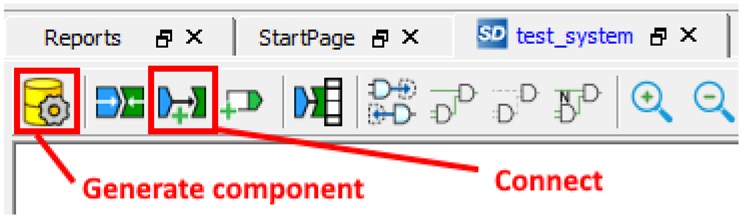
Top Module(root): MakerDemo
Active Synthesis Implementation: synthesis

Design Entry
SmartDesign Check

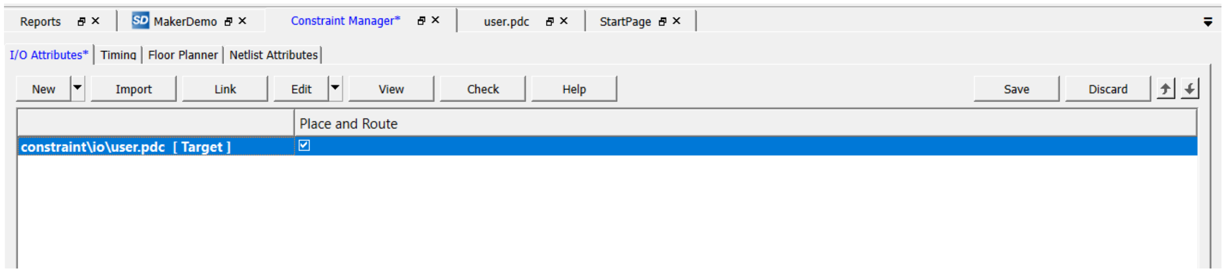
Fam: SmartFusion2 Part: M2S010-TQ144 Verilog

17. Click on the minus boxes on the right of the block to collapse the ports into buses.

18. Select the **GPIO_FABRIC** interface, right-click and select **Promote to Top Level** in the pop-up menu. This will create an output port called **GPIO_FABRIC** and connect the MSS design to the port.
19. Expand the **FAB_CCC_PINS** and **INIT_PINS** interfaces.
20. Right-click **MSS_READY** and click **Mark Unused** in the pop-up-menu.
21. Right-click **FAB_CCC_GL0** and click **Mark Unused** in the pop-up-menu.
22. Right-click **FAB_CCC_LOCK** and click **Mark Unused** in the pop-up-menu.
23. Right-click **INIT_DONE** and click **Mark Unused** in the pop-up-menu.
24. Make a connection between the **POWER_ON_RESET_N** and **FAB_RESET_N** pins of the MSS using the **Connection mode** button in SmartDesign (alternative: **Right-click > Connection mode** within the SmartDesign window).
25. Save the design, then click the **Generate component** button in SmartDesign (alternative: **Right-click > Generate component** within the SmartDesign window).

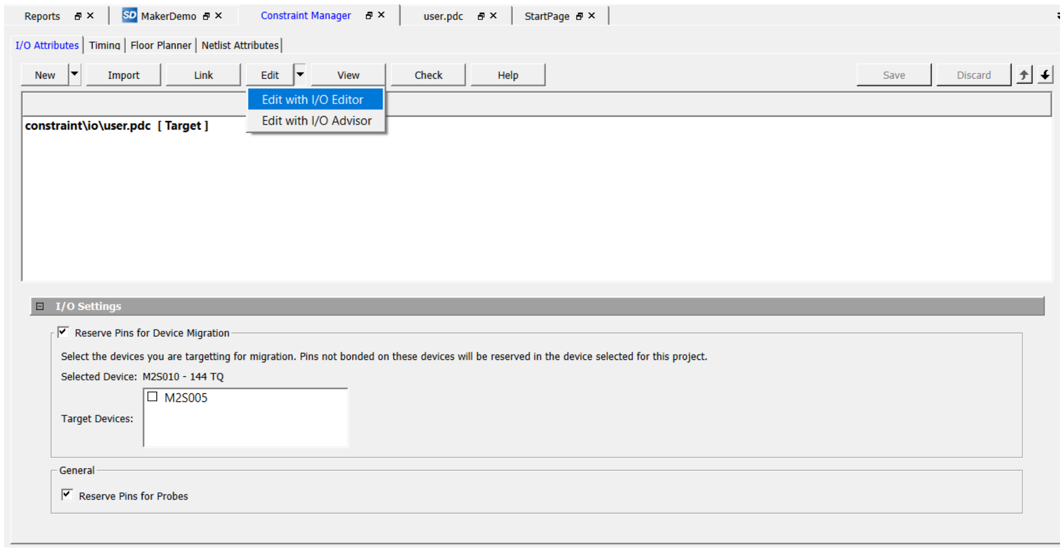


26. In the **Design Flow** pane to the left of the Libero window, double-click **Synthesize**.
27. You will see a number of warnings in the messages that refer to unused components and can be safely ignored.
28. When synthesis has completed, double-click **Manage Constraints** in the Design Flow pane.
29. Choose **New > Create New I/O Constraint from Root Module** and give the constraint a name (the one in the example design was given the name user) and click **OK**. Libero will auto-generate a constraint file populated with the top-level ports from the block diagram.
30. Back in the Constraint Manager window, right-click the new constraint and click **Set as Target**.
31. Tick the tick-box under **Place and Route** for the constraint.



	Place and Route
constraint\io\user.pdc [Target]	☑

32. Select **Edit > Edit with I/O Editor**.



constraint\io\user.pdc [Target]

I/O Settings

☑ Reserve Pins for Device Migration

Select the devices you are targetting for migration. Pins not bonded on these devices will be reserved in the device selected for this project.

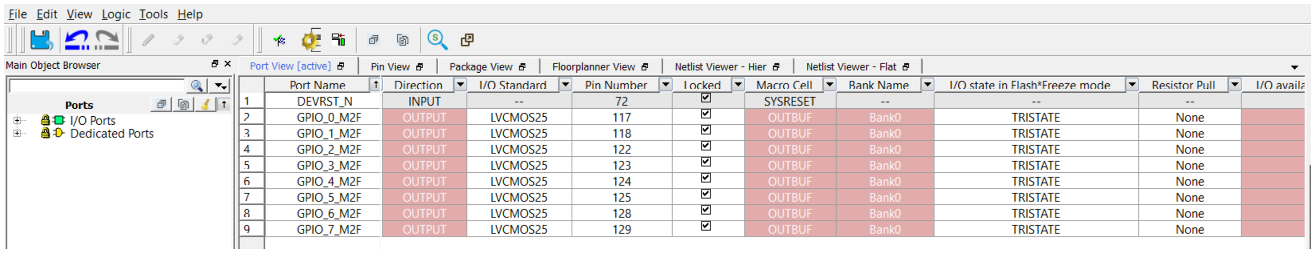
Selected Device: M2S010 - 144 TQ

Target Devices: ☐ M2S005

General

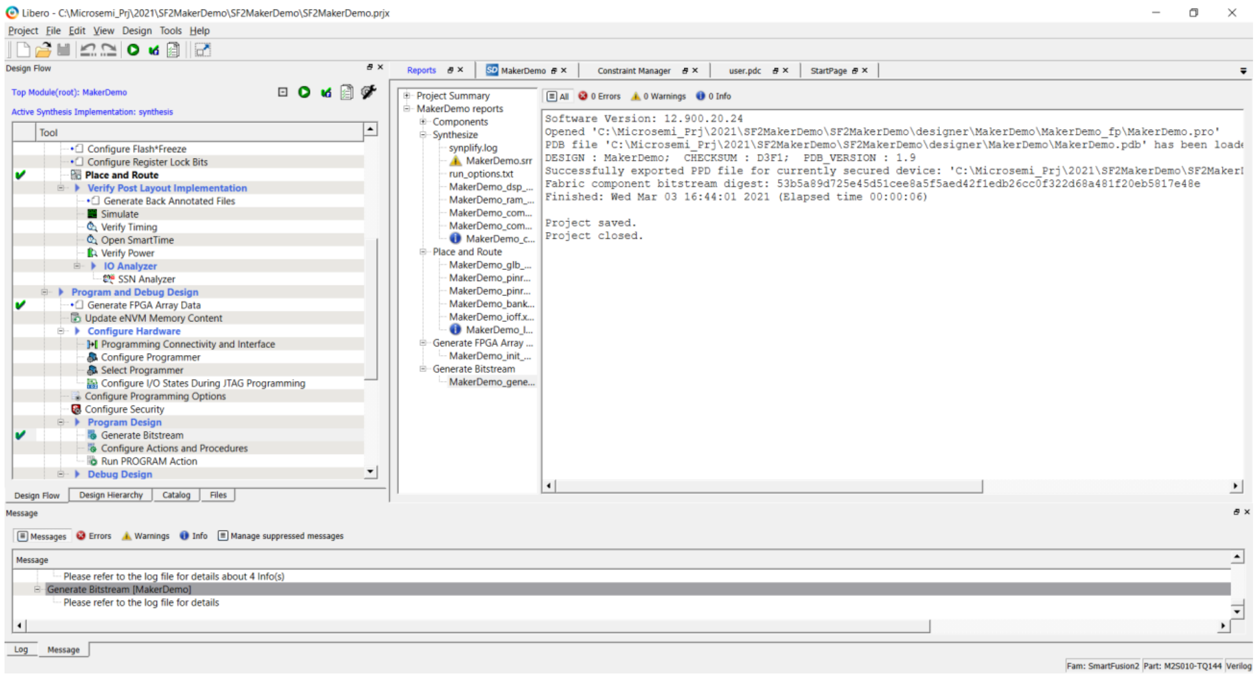
☑ Reserve Pins for Probes

33. You can edit the pin assignments to signals in the I/O Editor. Make the following settings in the I/O Editor window:



	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name	I/O state in Flash*Freeze mode	Resistor Pull	I/O availa
1	DEVRST_N	INPUT	--	72	✓	SYSRESET	--	--	--	
2	GPIO_0_M2F	OUTPUT	LVCMOS25	117	✓	OUTBUF	Bank0	TRISTATE	None	
3	GPIO_1_M2F	OUTPUT	LVCMOS25	118	✓	OUTBUF	Bank0	TRISTATE	None	
4	GPIO_2_M2F	OUTPUT	LVCMOS25	122	✓	OUTBUF	Bank0	TRISTATE	None	
5	GPIO_3_M2F	OUTPUT	LVCMOS25	123	✓	OUTBUF	Bank0	TRISTATE	None	
6	GPIO_4_M2F	OUTPUT	LVCMOS25	124	✓	OUTBUF	Bank0	TRISTATE	None	
7	GPIO_5_M2F	OUTPUT	LVCMOS25	125	✓	OUTBUF	Bank0	TRISTATE	None	
8	GPIO_6_M2F	OUTPUT	LVCMOS25	128	✓	OUTBUF	Bank0	TRISTATE	None	
9	GPIO_7_M2F	OUTPUT	LVCMOS25	129	✓	OUTBUF	Bank0	TRISTATE	None	

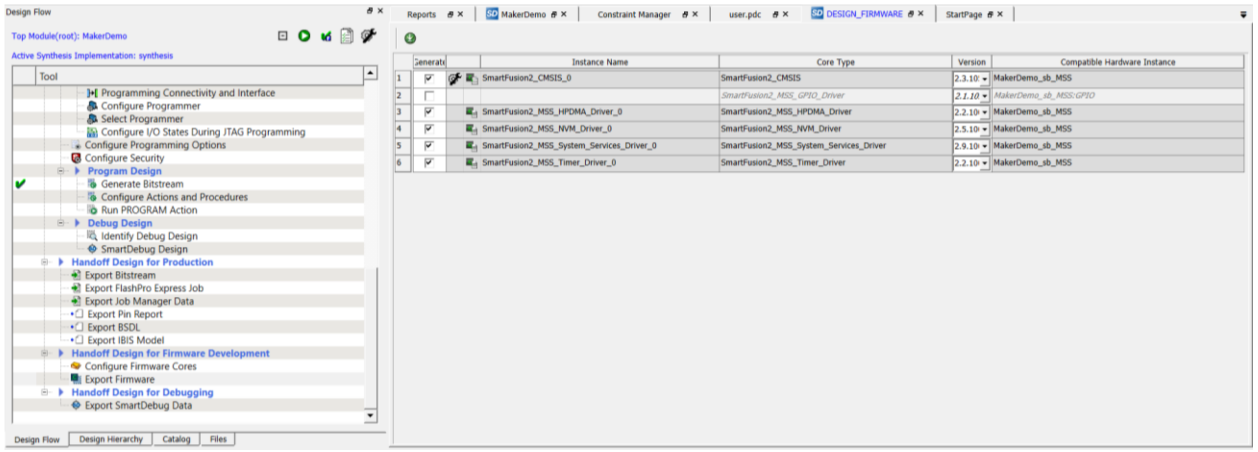
34. Save the constraints and close the I/O Editor. Select Yes to update the pdc file.
35. Back in the Design Flow pane, double-click **Place and Route**.
36. When Place and Route has completed, run **Generate FPGA Array Data**.
37. After the FPGA array data is generated, run **Generate Bitstream**.



```
Software Version: 12.900.20.24
Opened 'C:\Microsemi_Prj\2021\SF2MakerDemo\SF2MakerDemo\designer\MakerDemo\MakerDemo_fp\MakerDemo.pro'
PDB file 'C:\Microsemi_Prj\2021\SF2MakerDemo\SF2MakerDemo\designer\MakerDemo\MakerDemo.pdb' has been load
DESIGN : MakerDemo;  CHECKSUM : D3F1;  PDB_VERSION : 1.9
Successfully exported PPD file for currently secured device: 'C:\Microsemi_Prj\2021\SF2MakerDemo\SF2Maker
Fabric component bitstream digest: 53b5a89d725e45d51cee8a5f5aed42f1edb26cc0f322d68a481f20eb5817e48e
Finished: Wed Mar 03 16:44:01 2021 (Elapsed time 00:00:06)

Project saved.
Project closed.
```

```
Please refer to the log file for details about 4 Info(s)
Generate Bitstream [MakerDemo]
Please refer to the log file for details
```

38. Near the bottom of the Design Flow pane, select **Configure Firmware Cores**.
39. Make sure all the available firmware is selected for generation. There should be one for each peripheral enabled at the beginning. Libero will ask to download any cores that are not already present on the computer.

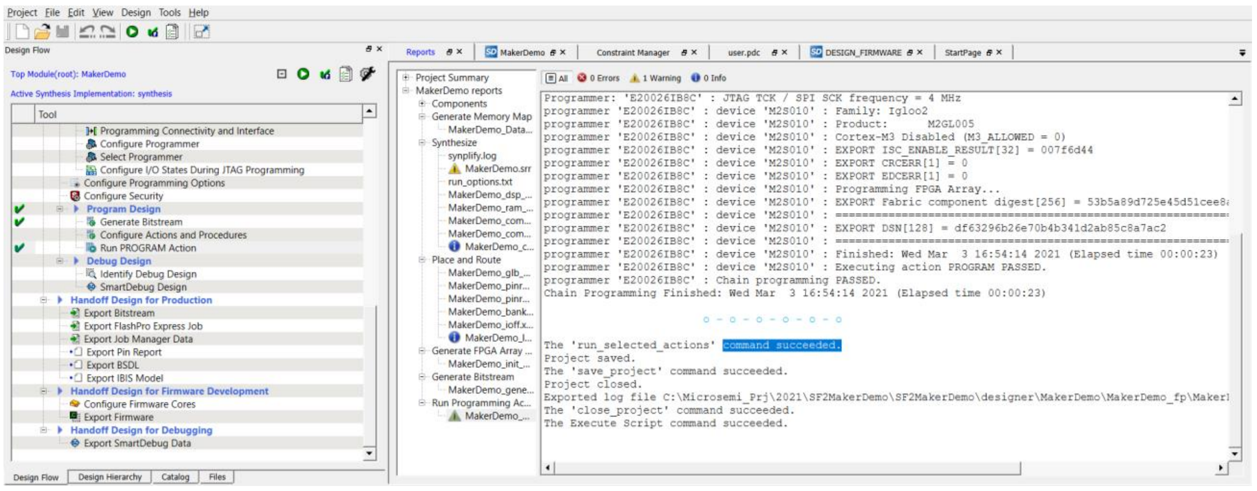


	Generate	Instance Name	Core Type	Version	Compatible Hardware Instance
1	✓	SmartFusion2_CMSIS_0	SmartFusion2_CMSIS	2.3.10	MakerDemo_sb_MSS
2			*SmartFusion2_MSS_GPIO_Driver*	2.1.10	*MakerDemo_sb_MSS:GPIO*
3	✓	SmartFusion2_MSS_HPDMA_Driver_0	SmartFusion2_MSS_HPDMA_Driver	2.2.10	MakerDemo_sb_MSS
4	✓	SmartFusion2_MSS_NVM_Driver_0	SmartFusion2_MSS_NVM_Driver	2.5.10	MakerDemo_sb_MSS
5	✓	SmartFusion2_MSS_System_Services_Driver_0	SmartFusion2_MSS_System_Services_Driver	2.9.10	MakerDemo_sb_MSS
6	✓	SmartFusion2_MSS_Timer_Driver_0	SmartFusion2_MSS_Timer_Driver	2.2.10	MakerDemo_sb_MSS

40. From the Design Flow pane, choose **Export Firmware**. Choose a location for the export and set the Software IDE to **SoftConsole 4.0**. This automatically selects the **Export hardware configuration and firmware drivers** setting.

PROGRAMMING THE FPGA USING LIBERO

1. Connect the M2S010-MKR-KIT to your computer using a USB cable.
2. Choose **Run PROGRAM Action** from the Design Flow pane to load the configuration onto the board.
3. Make sure the report appearing in the **Reports** window returns a success.

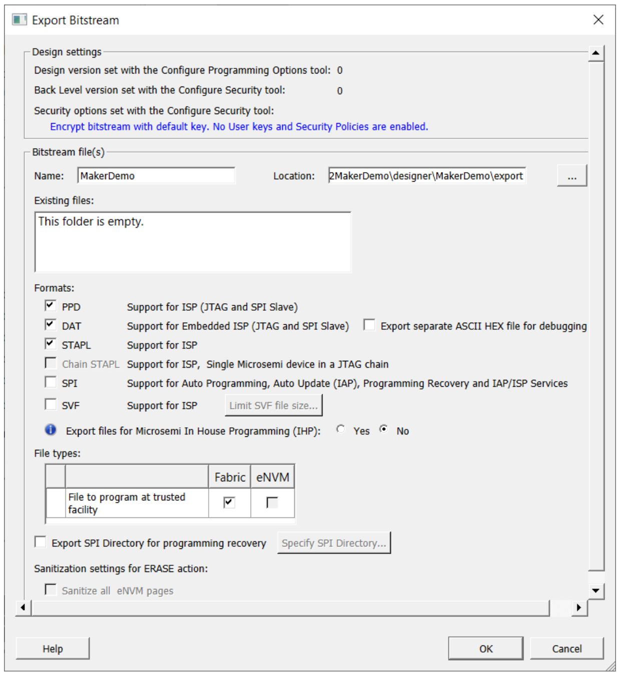


```
Programmer: 'E20026IB8C' : JTAG TCK / SPI SCK frequency = 4 MHz
programmer 'E20026IB8C' : device 'M2S010' : Family: Igloo2
programmer 'E20026IB8C' : device 'M2S010' : Product:      M2GL005
programmer 'E20026IB8C' : device 'M2S010' : Cortex-M3 Disabled (M3_ALLOWED = 0)
programmer 'E20026IB8C' : device 'M2S010' : EXPORT ISC_ENABLE_RESULT[32] = 007f6d44
programmer 'E20026IB8C' : device 'M2S010' : EXPORT CRCERR[1] = 0
programmer 'E20026IB8C' : device 'M2S010' : EXPORT EDCERR[1] = 0
programmer 'E20026IB8C' : device 'M2S010' : Programming FPGA Array...
programmer 'E20026IB8C' : device 'M2S010' : EXPORT Fabric component digest[256] = 53b5a89d725e45d51cee8
programmer 'E20026IB8C' : device 'M2S010' : ==============================================================
programmer 'E20026IB8C' : device 'M2S010' : EXPORT DSN[128] = df63296b26e70b4b341d2ab85c8a7ac2
programmer 'E20026IB8C' : device 'M2S010' : ==============================================================
programmer 'E20026IB8C' : device 'M2S010' : Finished: Wed Mar  3 16:54:14 2021 (Elapsed time 00:00:23)
programmer 'E20026IB8C' : device 'M2S010' : Executing action PROGRAM PASSED.
programmer 'E20026IB8C' : Chain programming PASSED.
Chain Programming Finished: Wed Mar  3 16:54:14 2021 (Elapsed time 00:00:23)

                 o - o - o - o - o - o

The 'run_selected_actions' command succeeded.
Project saved.
The 'save_project' command succeeded.
Project closed.
Exported log file C:\Microsemi_Prj\2021\SF2MakerDemo\SF2MakerDemo\designer\MakerDemo\MakerDemo_fp\Maker
The 'close_project' command succeeded.
The Execute Script command succeeded.
```

EXPORTING THE BITSTREAM FOR PROGRAMMING USING FLASHPRO

Follow these steps:

1. In the Design Flow pane, double-click **Export Bitstream** under **Handoff Design for Production**.
2. The Export Bitstream window will appear. Select a location where the file should be exported to and make sure **STAPL** and **Fabric** are ticked. You may optionally add a suffix for the name by typing in some text in the text box.



Export Bitstream

Design settings

Design version set with the Configure Programming Options tool: 0

Back Level version set with the Configure Security tool: 0

Security options set with the Configure Security tool:

Encrypt bitstream with default key. No User keys and Security Policies are enabled.

Bitstream file(s)

Name: MakerDemo   Location: 2MakerDemo\designer\MakerDemo\export

Existing files:

This folder is empty.

Formats:

☑ PPD	Support for ISP (JTAG and SPI Slave)	
☑ DAT	Support for Embedded ISP (JTAG and SPI Slave)	☐ Export separate ASCII HEX file for debugging
☑ STAPL	Support for ISP	
☐ Chain STAPL	Support for ISP, Single Microsemi device in a JTAG chain	
☐ SPI	Support for Auto Programming, Auto Update (IAP), Programming Recovery and IAP/ISP Services	
☐ SVF	Support for ISP	Limit SVF file size...

Export files for Microsemi In House Programming (IHP): Yes [✓ No]

File types:

	Fabric	eNVM
File to program at trusted facility	☑	☐

☐ Export SPI Directory for programming recovery   Specify SPI Directory...

Sanitization settings for ERASE action:

☐ Sanitize all eNVM pages

3. Click OK. After a while, the bitstream is exported into a STAPL (.stp) file to the location of your choice. This STAPL file can be used by anyone to program the bitstream using FlashPro.
4. Verify the report appearing in the Reports window returns a success.

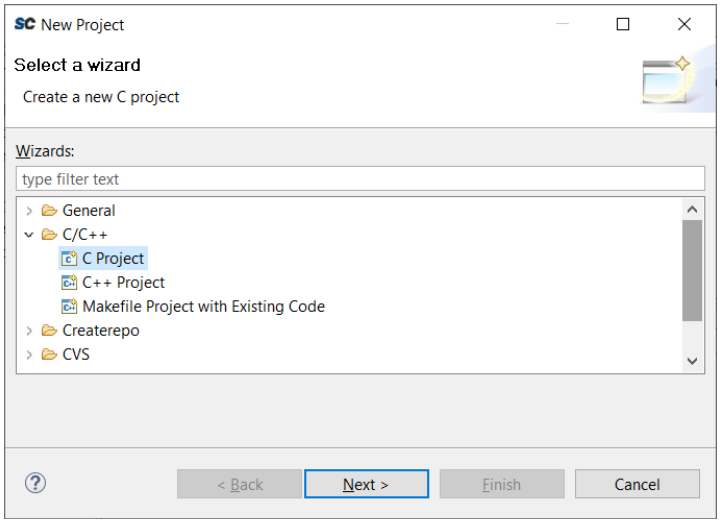
III. CREATE SOFTWARE USING SOFTCONSOLE

Using the MSS system generated in the steps above, we can now proceed to writing some firmware to light the LEDs. We will use the firmware generated from Libero in the steps above along with some custom code to blink the LEDs on the M2S010-MKR-KIT.

CREATING YOUR FIRST SOFTCONSOLE PROJECT

Follow these steps:

1. Open SoftConsole and choose a convenient directory as the workspace.
2. In SoftConsole, select **File > New > Project...**
3. In the window that opens, select **C Project** and click **Next**.



New Project

Select a wizard

Create a new C project

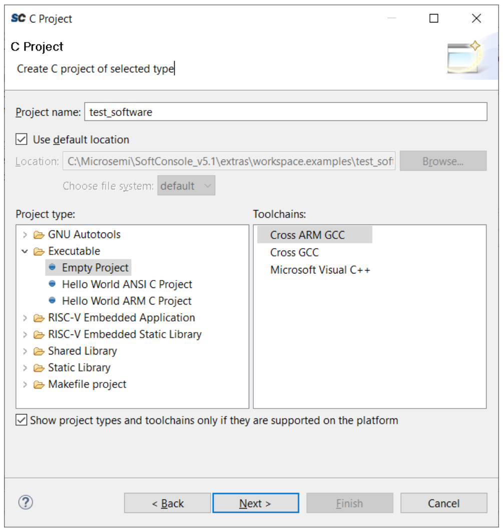
Wizards:

type filter text

- General
- C/C++
  - C Project
  - C++ Project
  - Makefile Project with Existing Code
- Createrepo
- CVS

< Back | Next > | Finish | Cancel

4. Name your software test_software and choose a location for the project. Click **Next** to create an Empty project using ARM Cross-GCC toolchain.



C Project

C Project

Create C project of selected type

Project name: test_software

☑ Use default location

Location: C:\Microsemi\SoftConsole_v5.1\extras\workspace.examples\test_sof

Browse...

Choose file system: default

Project type:

- GNU Autotools
- Executable
  - Empty Project
  - Hello World ANSI C Project
  - Hello World ARM C Project
- RISC-V Embedded Application
- RISC-V Embedded Static Library
- Shared Library
- Static Library
- Makefile project

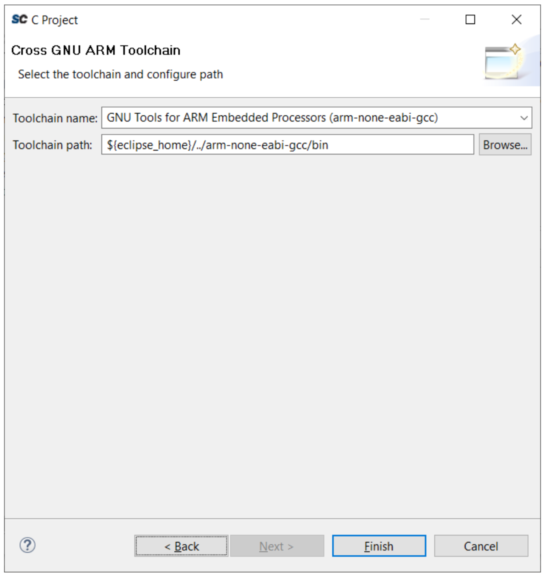
Toolchains:

- Cross ARM GCC
- Cross GCC
- Microsoft Visual C++

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

< Back | Next > | Finish | Cancel

5. Click **Next** to add the default Debug and Release configurations to the project.
6. Click **Finish** to accept the default Toolchain settings.


C Project

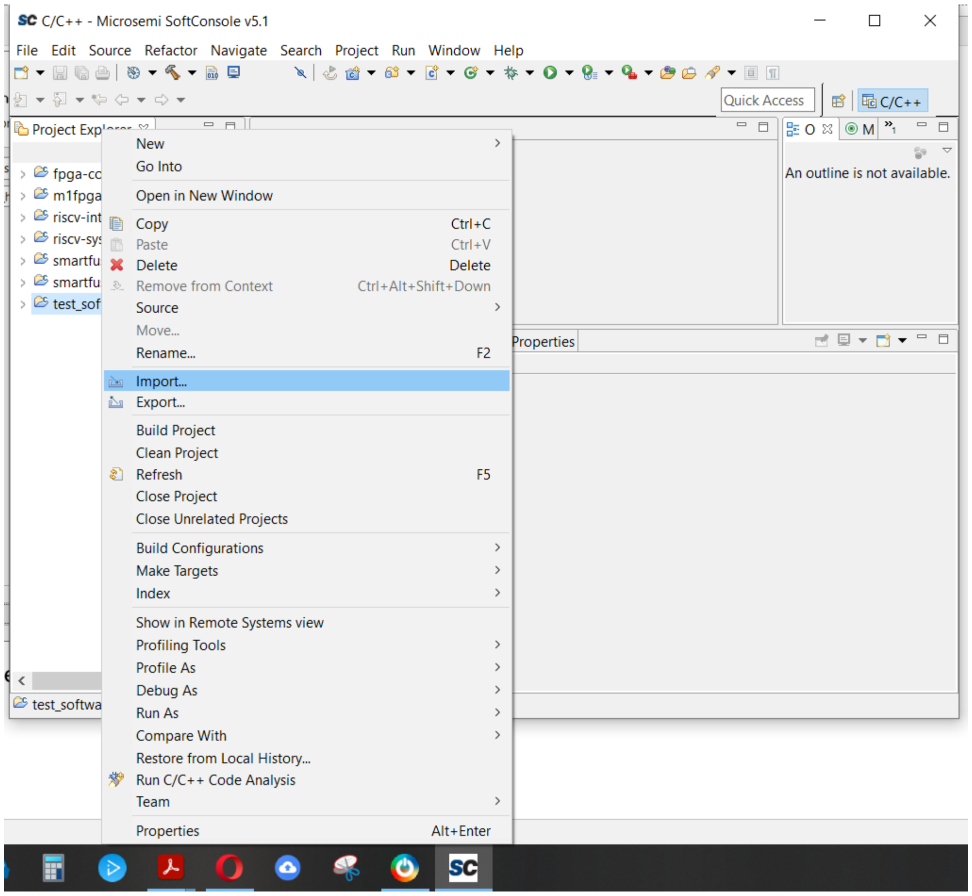
Cross GNU ARM Toolchain

Select the toolchain and configure path

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

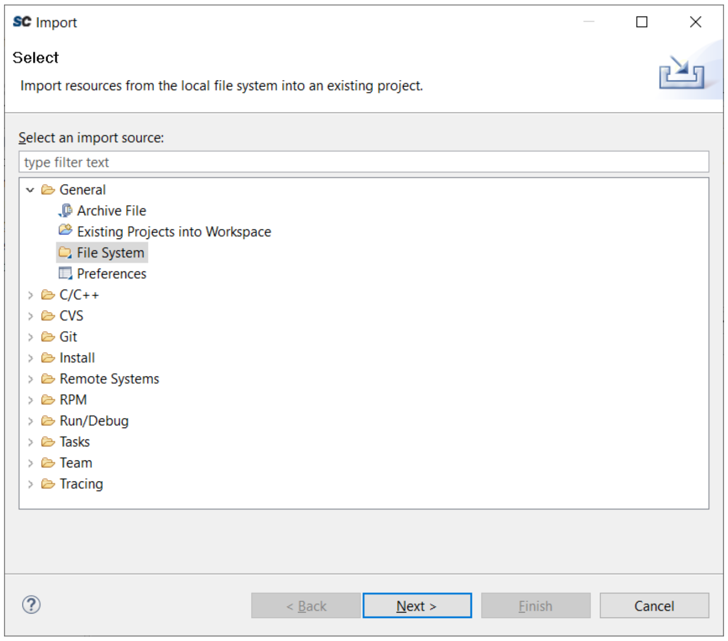
Toolchain path: ${eclipse_home}/../arm-none-eabi-gcc/bin

7. Your new project is created. Right-click it in the **Project Explorer** pane and click **Import...**


C/C++ - Microsemi SoftConsole v5.1

An outline is not available.

8. In the Import window that opens, under General select **File System** and click **Next**.



Import

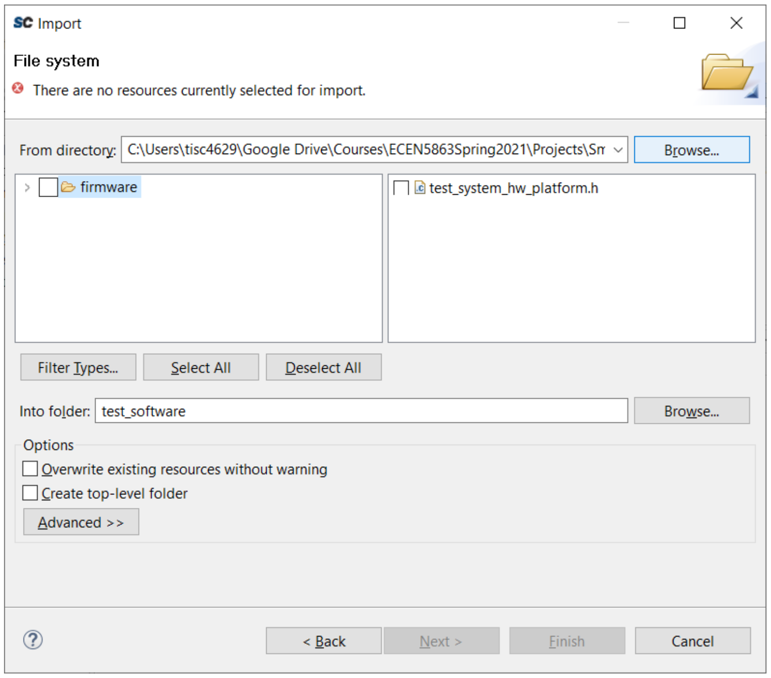
Select

Import resources from the local file system into an existing project.

Select an import source:

type filter text

9. Unzip M2S010-MKR-KIT_FirstProj-master.zip. Browse to the firmware/ directory under the M2S010-MKR-KIT_FirstProj-master project directory, select it and click **OK**.



Import

File system

There are no resources currently selected for import.

From directory: C:\Users\tisc4629\Google Drive\Courses\ECEN5863Spring2021\Projects\Sm

firmware

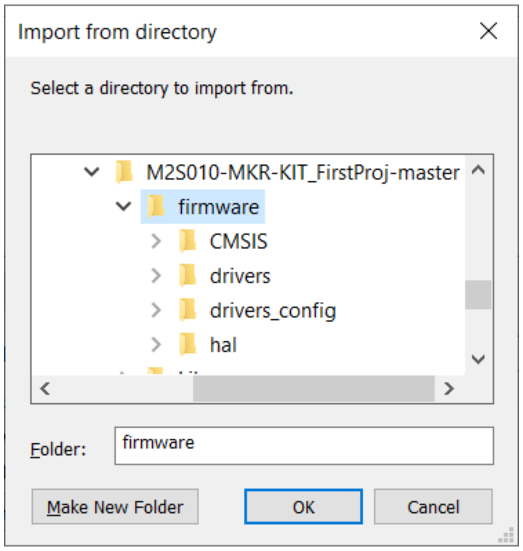
test_system_hw_platform.h

Into folder: test_software

Options

Overwrite existing resources without warning

Create top-level folder

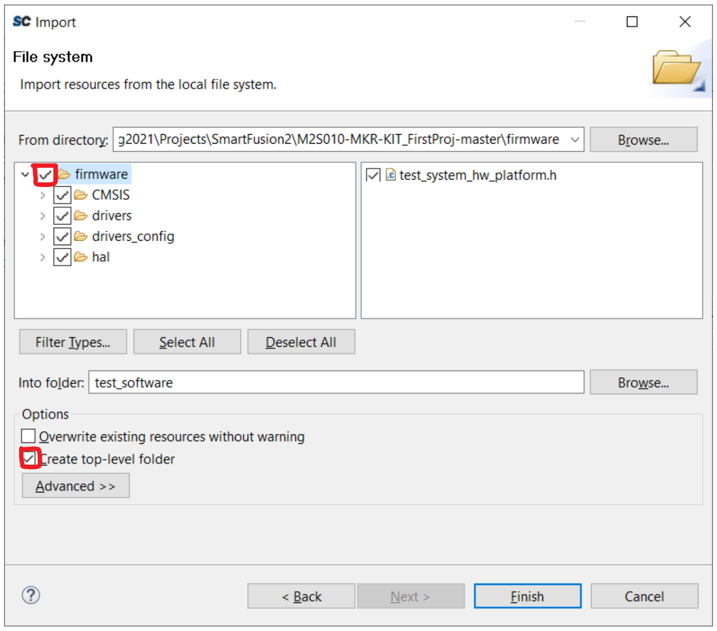

Import from directory

Select a directory to import from.

M2S010-MKR-KIT_FirstProj-master
firmware
CMSIS
drivers
drivers_config
hal

Folder: firmware

10. Click the **firmware** folder tick-box in the Import window, click the **Create top-level folder** tick-box and click **Finish**.


Import

File system

Import resources from the local file system.

From directory: g2021\Projects\SmartFusion2\M2S010-MKR-KIT_FirstProj-master\firmware

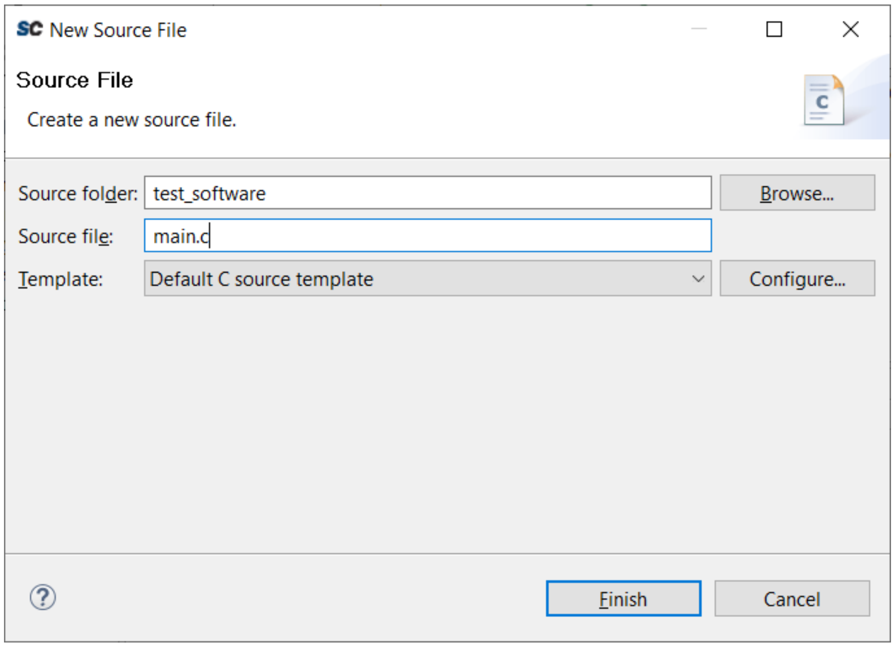
[✓] firmware
[✓] CMSIS
[✓] drivers
[✓] drivers_config
[✓] hal

[✓] test_system_hw_platform.h

Into folder: test_software

Options
[ ] Overwrite existing resources without warning
[✓] Create top-level folder

11. Your **test_software** project is now ready for writing some code to blink the LEDs. It is time to create a C file to contain our main() function. In the main menu of SoftConsole, Select **File > New > Source file**.
12. In the window that appears, give the file the name main.c and make sure it lies under the "**test_software**" folder. Click **Finish**.


New Source File

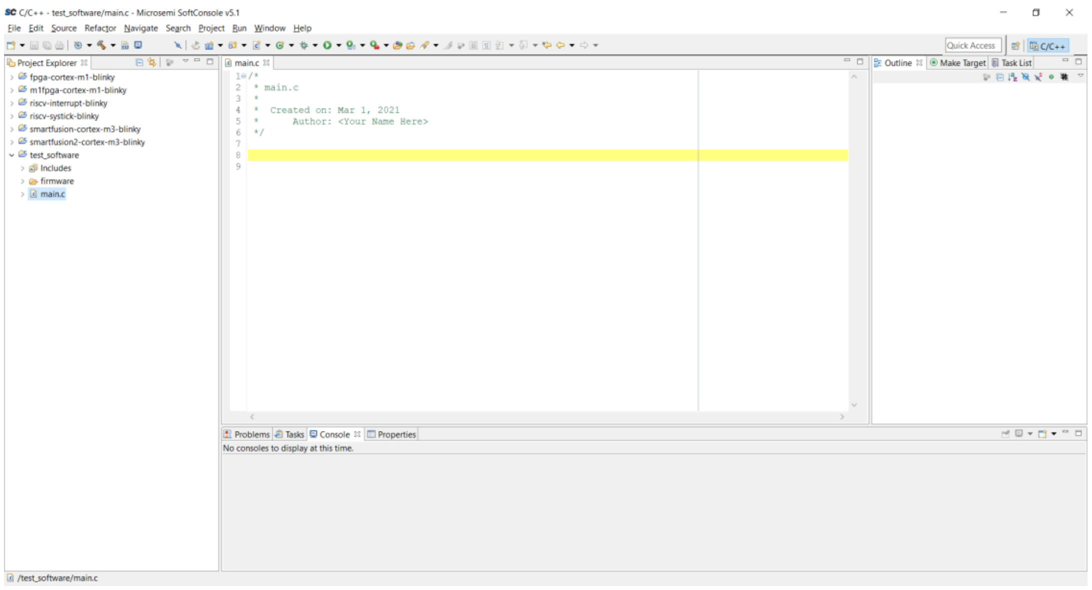
Source File

Create a new source file.

Source folder: test_software

Source file: main.c

Template: Default C source template


C/C++ - test_software/main.c - Microsemi SoftConsole v5.1

```
/*
 * main.c
 *
 *  Created on: Mar 1, 2021
 *      Author: <Your Name Here>
 */
```

No consoles to display at this time.

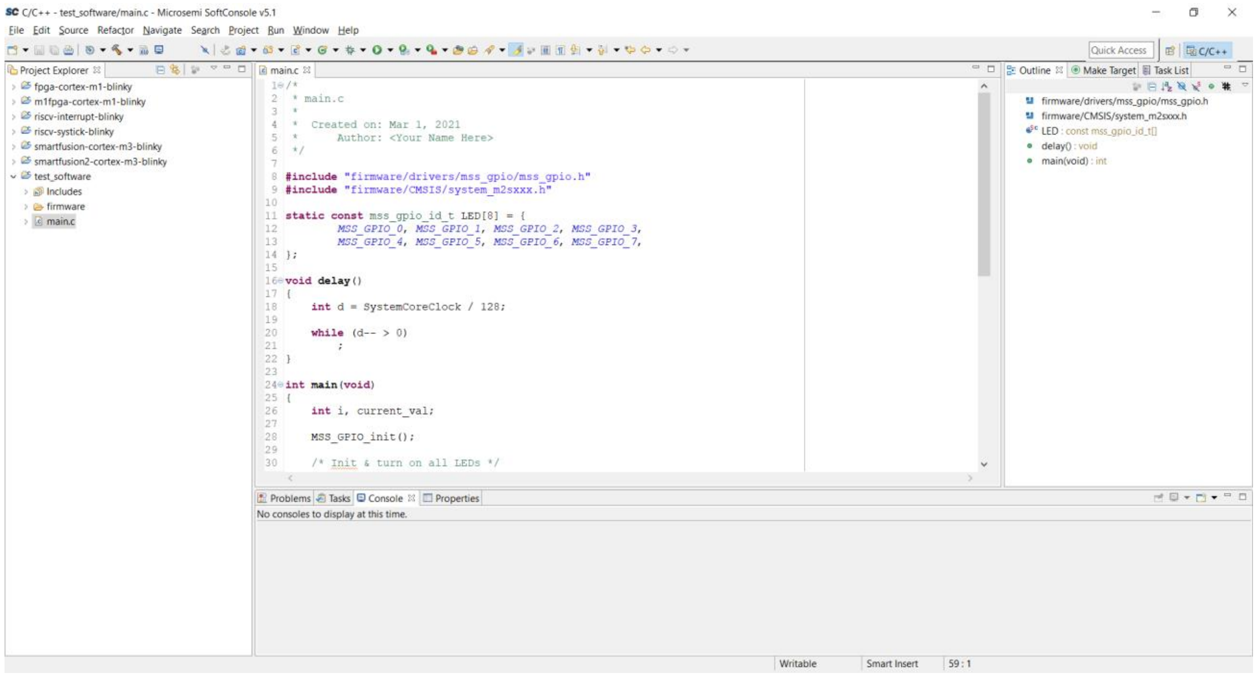
/test_software/main.c

13. The new source file should open. If it doesn't, open it by double-clicking it in the Project Explorer pane. Copy the following code to the file (identical to that in the main.c source file from the repository):

```
#include "firmware/drivers/mss_gpio/mss_gpio.h"
#include "firmware/CMSIS/system_m2sxxx.h"

static const mss_gpio_id_t LED[8] = {
             MSS_GPIO_0, MSS_GPIO_1, MSS_GPIO_2, MSS_GPIO_3,
             MSS_GPIO_4, MSS_GPIO_5, MSS_GPIO_6, MSS_GPIO_7,
};
```

```
void delay()
{
	int d = SystemCoreClock / 128;

	while (d-- > 0)
		;
}

int main(void)
{
	int i, current_val;

	MSS_GPIO_init();

	/* Init & turn on all LEDs */
	for (i = 0; i < 8; i++) {
		MSS_GPIO_config(LED[i], MSS_GPIO_OUTPUT_MODE);
		MSS_GPIO_set_output(LED[i], 0);
	}
	delay();
	delay();
	delay();
	delay();

	/* Blink all LEDs on start up */
	for (i = 0; i < 8; i++) {
		for (current_val = 0; current_val < 8; ++current_val)
			MSS_GPIO_set_output(LED[current_val], 1);
		delay();
		for (current_val = 0; current_val < 8; ++current_val)
			MSS_GPIO_set_output(LED[current_val], 0);
		delay();
	}

	/* Sequence-blink */
	for (;;) {
```

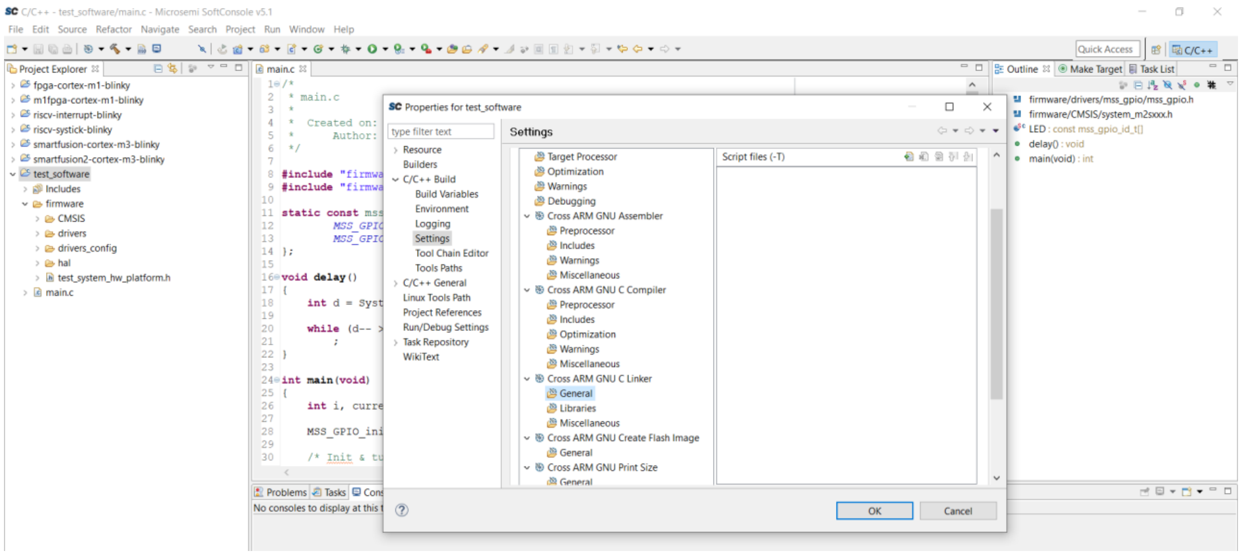
```
                for (i = 0; i < 8; i++) {
                        current_val = (MSS_GPIO_get_outputs() & (1 << LED[i])) ? 1
: 0;
                        MSS_GPIO_set_output(LED[i], current_val ^ 1);
                        delay();
                }
        }
}
```

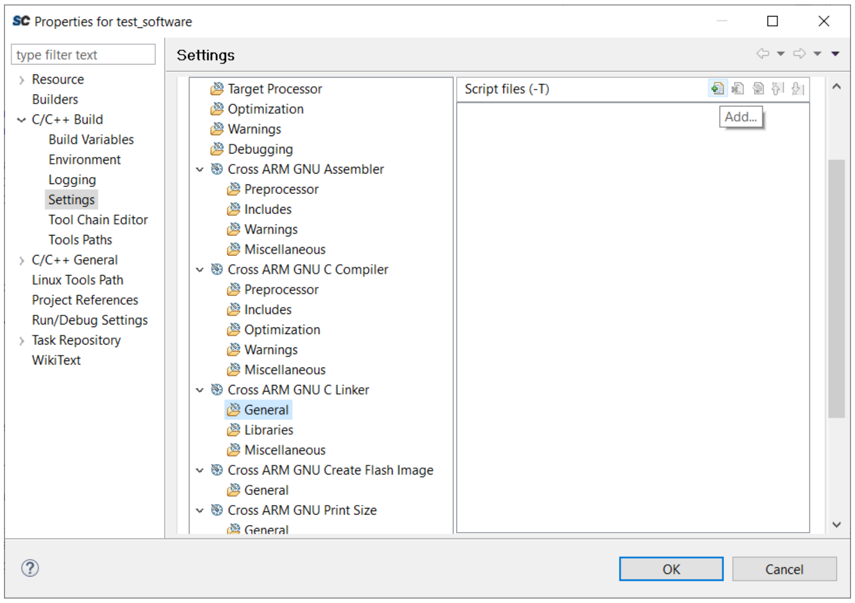


```
/*
 * main.c
 *
 *  Created on: Mar 1, 2021
 *      Author: <Your Name Here>
 */

#include "firmware/drivers/mss_gpio/mss_gpio.h"
#include "firmware/CMSIS/system_m2sxxx.h"

static const mss_gpio_id_t LED[8] = {
        MSS_GPIO_0, MSS_GPIO_1, MSS_GPIO_2, MSS_GPIO_3,
        MSS_GPIO_4, MSS_GPIO_5, MSS_GPIO_6, MSS_GPIO_7,
};

void delay()
{
    int d = SystemCoreClock / 128;

    while (d-- > 0)
        ;
}

int main(void)
{
    int i, current_val;

    MSS_GPIO_init();

    /* Init & turn on all LEDs */
```

No consoles to display at this time.

14. Right-click the project in the Project Explorer pane and click **Properties**.
15. In the Properties window that opens up, navigate to **C/C++ Build > Settings**.
16. In the **Tool Settings** tab, navigate to **GNU ARM Cross C Linker > General** and click the **Add** button under **Script files (-T)**:


C/C++ - test_software/main.c - Microsemi SoftConsole v5.1
Project Explorer
test_software
main.c
Properties for test_software
Settings
Script files (-T)
OK
Cancel



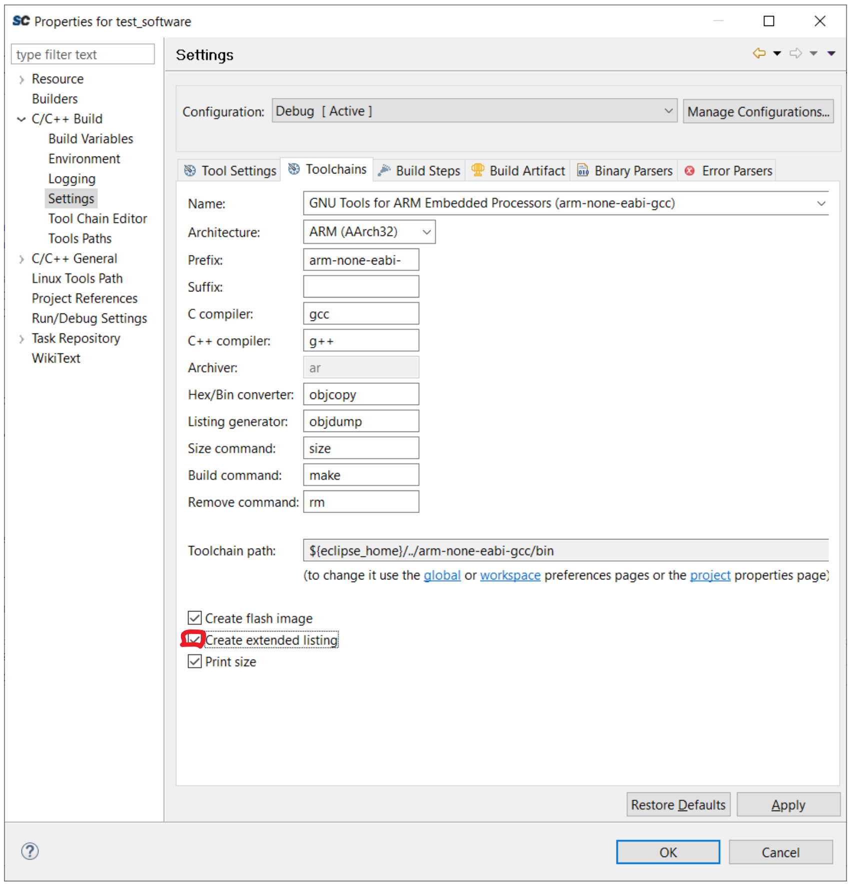
17. In the window that opens up, add the relative path to the linker script:

    ../firmware/CMSIS/startup_gcc/debug-in-microsemi-smartfusion2-esram.ld

18. Under **GNU ARM Cross C Linker > Miscellaneous**, tick the **Use newlib-nano (--specs=nano.specs)** tick-box.
19. Under **GNU ARM Cross C Compiler > Miscellaneous**, fill in the following in the **Other compiler flags** field:

    --specs=cmsis.specs

20. In the **Toolchains** tab, tick **Create extended listing**.



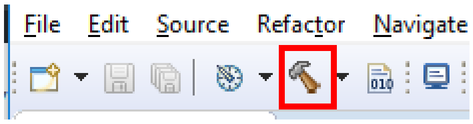
Configuration: Debug [Active]

Name: GNU Tools for ARM Embedded Processors (arm-none-eabi-gcc)
Architecture: ARM (AArch32)
Prefix: arm-none-eabi-
Suffix:
C compiler: gcc
C++ compiler: g++
Archiver: ar
Hex/Bin converter: objcopy
Listing generator: objdump
Size command: size
Build command: make
Remove command: rm

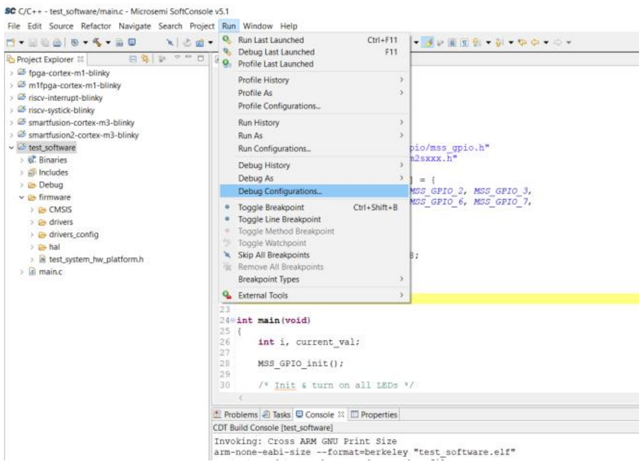
Toolchain path: ${eclipse_home}/../arm-none-eabi-gcc/bin
(to change it use the global or workspace preferences pages or the project properties page)

- [✓] Create flash image
- [✓] Create extended listing
- [✓] Print size

21. Click **Apply and OK**.
22. Build the project by hitting **Ctrl+B** on your keyboard, or clicking the build button. The project should build successfully.



23. In the file menu, click **Run > Debug Configurations...**.

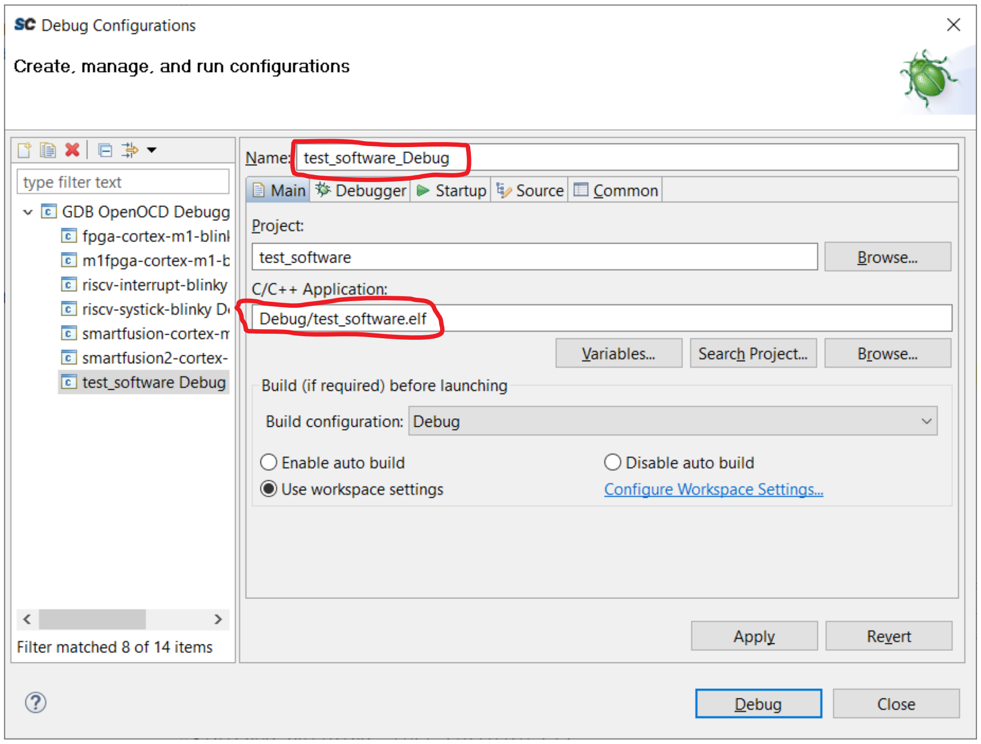


```
pio/mss_gpio.h"
m2sxxx.h"

] = {
MSS_GPIO_2, MSS_GPIO_3,
MSS_GPIO_6, MSS_GPIO_7,

B;

int main(void)
{
    int i, current_val;

    MSS_GPIO_init();

    /* Init & turn on all LEDs */
```

CDT Build Console [test_software]

```
Invoking: Cross ARM GNU Print Size
arm-none-eabi-size --format=berkeley "test_software.elf"
```

24. Double-click **GDB OpenOCD Debugging**. In the **Main** tab, name the new configuration test_software_Debug and replace the backslash with a forward slash in the **C/C++ Application** text box (this is to make the project portable on Linux).



Debug Configurations

Create, manage, and run configurations

Name: test_software_Debug

Project: test_software

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

Build (if required) before launching

Build configuration: Debug

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

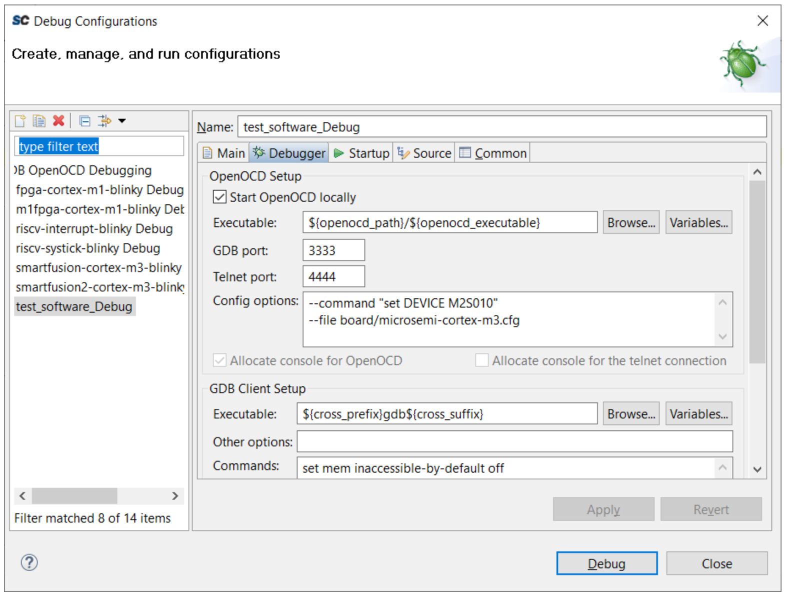
Filter matched 8 of 14 items

25. In the **Debugger** tab, add the following text under **Config options**:

--command "set DEVICE M2S010"

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

Delete all other commands present in this box.



Debug Configurations

Create, manage, and run configurations

Name: test_software_Debug

OpenOCD Setup

[✓] Start OpenOCD locally

Executable: ${openocd_path}/${openocd_executable}

GDB port: 3333

Telnet port: 4444

Config options:
```
--command "set DEVICE M2S010"
--file board/microsemi-cortex-m3.cfg
```

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

GDB Client Setup

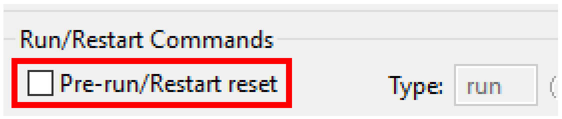
Executable: ${cross_prefix}gdb${cross_suffix}

Other options:

Commands: set mem inaccessible-by-default off

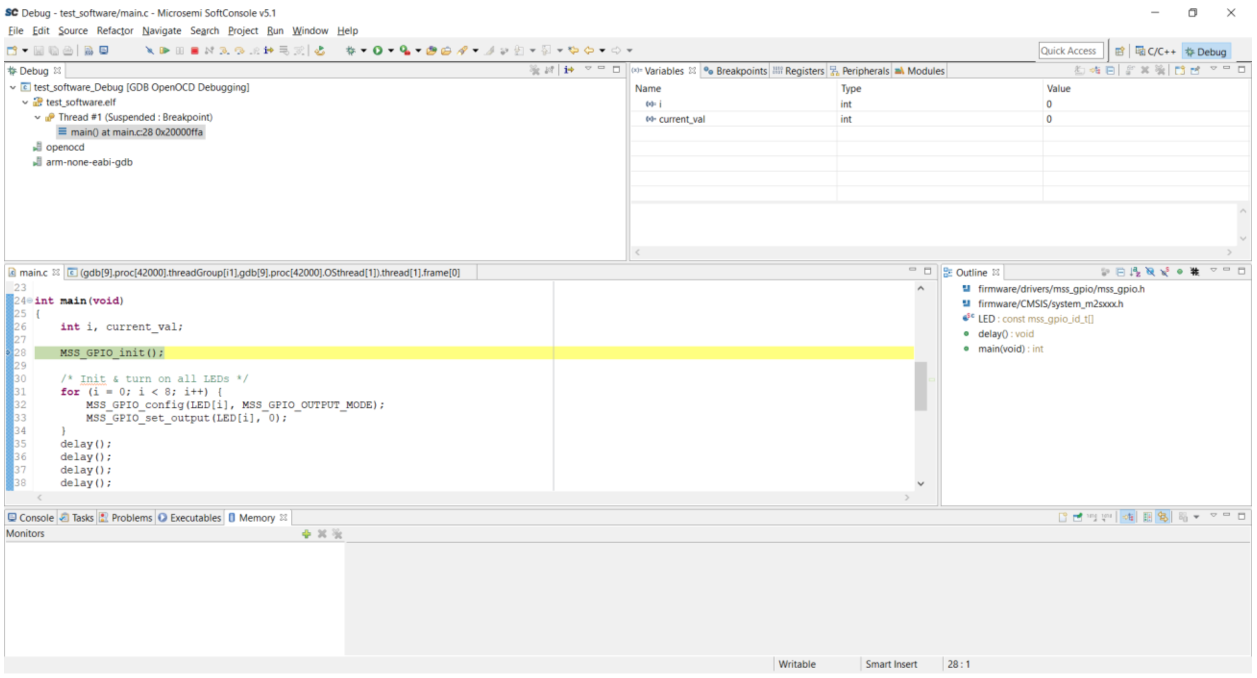
Filter matched 8 of 14 items

26. In the **Startup** tab, scroll down to the **Run/Restart Commands** section and make the **Pre-run/Restart reset** tick-box is not checked.


Run/Restart Commands

☐ Pre-run/Restart reset   Type: run

27. Plug the M2S010-MKR-KIT into a USB port and click **Debug**.
28. The **Debug** window should appear (Click Yes if it asks) and a breakpoint is set on entry to the main() function.



```
test_software_Debug [GDB OpenOCD Debugging]
  test_software.elf
    Thread #1 (Suspended : Breakpoint)
      main() at main.c:28 0x20000ffa
  openocd
  arm-none-eabi-gdb
```

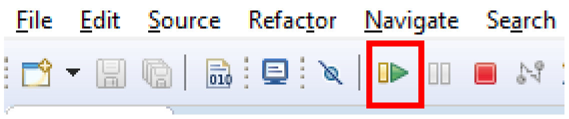
Name	Type	Value
i	int	0
current_val	int	0

main.c — (gdb[9].proc[42000].threadGroup[i1],gdb[9].proc[42000].OSthread[1]).thread[1].frame[0]

```
int main(void)
{
    int i, current_val;

    MSS_GPIO_init();

    /* Init & turn on all LEDs */
    for (i = 0; i < 8; i++) {
        MSS_GPIO_config(LED[i], MSS_GPIO_OUTPUT_MODE);
        MSS_GPIO_set_output(LED[i], 0);
    }
    delay();
    delay();
    delay();
    delay();
```

```
firmware/drivers/mss_gpio/mss_gpio.h
firmware/CMSIS/system_m2sxxx.h
LED : const mss_gpio_id_t[]
delay() : void
main(void) : int
```

29. Click the **Resume** button to run the program.



29. The LEDs should now be blinking. At first all LEDs will blink simultaneously, then they will be turned on and off sequentially.
30. Play around in the code to see changes taking effect. For example, change the division factor in the delay() function, build and debug the project again. Note that the build step is necessary prior to debugging.

BONUS (10 points): Change the hardware and software so that the light pattern switches direction when one of the user buttons is pressed.

PROGRAMMING THE ENVM FROM SOFTCONSOLE

In order to have the firmware running on startup, it must be programmed into the eNVM (embedded non-volatile memory) of the SmartFusion2 FPGA. This is done by simply changing the linker script in step 17 above to the eNVM linker script, debug-in-microsemi-smartfusion2-envm.ld

The full process to do this follows:

1. Right-click the project in the Project Explorer pane and click **Properties**.
2. In the Properties window that opens up, navigate to **C/C++ Build > Settings**.
3. In the **Tool Settings** tab, navigate to **GNU ARM Cross C Linker > General** and click the **Add** button under **Script files (-T)**.
4. In the window that opens up, add the relative path to the linker script:

   ../firmware/CMSIS/startup_gcc/debug-in-microsemi-smartfusion2-envm.ld

   The next time debug runs, code will be downloaded to the eNVM and will start executing upon subsequent power-ups.

   NB: The linker script shown here is a demonstration for the example project. For your own project, a release build should be compiled and downloaded to the eNVM.