# Contents

1 Revision History .................................................. 1
  1.1 Revision 7.0 ..................................................... 1
  1.2 Revision 6.0 ..................................................... 1
  1.3 Revision 5.0 ..................................................... 1
  1.4 Revision 4.0 ..................................................... 1
  1.5 Revision 3.0 ..................................................... 1
  1.6 Revision 2.0 ..................................................... 1
  1.7 Revision 1.0 ..................................................... 1

2 Building a Cortex-M1 Processor Subsystem ....................... 2
  2.1 Requirements ................................................... 2
  2.2 Prerequisites ................................................... 3
  2.3 Creating a Cortex-M1 Processor Subsystem ....................... 3
    2.3.1 Creating a Libero SoC Project ................................ 4
    2.3.2 Creating a New SmartDesign Component .......................... 7
    2.3.3 Instantiating the IP Cores in SmartDesign ....................... 7
    2.3.4 Connecting IP Blocks in SmartDesign ............................ 15
    2.3.5 Generating SmartDesign Component .............................. 17
    2.3.6 Managing Timing Constraints ................................... 18
    2.3.7 Running Libero Design Flow ................................... 18

3 Creating User Application Using SoftConsole ..................... 29
  3.1 Creating a Cortex-M1 Project .................................... 29
  3.2 Downloading the Firmware Drivers ................................ 31
  3.3 Importing the Firmware Drivers .................................. 33
  3.4 Creating the main.c File ........................................ 34
  3.5 Configuring the Cortex-M1 Project ................................ 35
  3.6 Mapping Memory and Peripheral Addresses ........................ 40
  3.7 Setting the UART Baud Rate ..................................... 41
  3.8 Building the User Application in Release Mode ..................... 42
  3.9 Building In Debug Mode and Debugging the User Application .......... 44

4 Appendix 1: Running the TCL Script ................................ 51

5 Appendix 2: References ........................................... 52
<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Add Core to Vault</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Block Diagram</td>
<td>4</td>
</tr>
<tr>
<td>Figure 3</td>
<td>New Project Details</td>
<td>5</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Device Selection</td>
<td>5</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Device Settings</td>
<td>6</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Add Constraints Window</td>
<td>6</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Create New SmartDesign</td>
<td>7</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Instantiating PF_INIT_MONITOR</td>
<td>8</td>
</tr>
<tr>
<td>Figure 9</td>
<td>PF_CCC Clock Options</td>
<td>9</td>
</tr>
<tr>
<td>Figure 10</td>
<td>PF_CCC Output Clocks</td>
<td>9</td>
</tr>
<tr>
<td>Figure 11</td>
<td>CoreCORTEXM1 Configurator</td>
<td>10</td>
</tr>
<tr>
<td>Figure 12</td>
<td>CoreAHBLite Configurator</td>
<td>11</td>
</tr>
<tr>
<td>Figure 13</td>
<td>PF_SRAM_AHBL_AXI Configurator</td>
<td>12</td>
</tr>
<tr>
<td>Figure 14</td>
<td>CoreAPB3 Configurator</td>
<td>13</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Core GPIO Configurator</td>
<td>14</td>
</tr>
<tr>
<td>Figure 16</td>
<td>CortexM1_Subsystem Without Connections</td>
<td>14</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Connection Method</td>
<td>15</td>
</tr>
<tr>
<td>Figure 18</td>
<td>CortexM1_Subsystem With Connections</td>
<td>16</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Modify Memory Map Dialog Box- APB3</td>
<td>17</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Modify Memory Map Dialog Box- CoreAHBLite</td>
<td>17</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Build Hierarchy option</td>
<td>17</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Generate Component</td>
<td>18</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Derived Constraints</td>
<td>18</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Synthesis Completion</td>
<td>19</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Manage Constraints</td>
<td>19</td>
</tr>
<tr>
<td>Figure 26</td>
<td>I/O Attributes</td>
<td>19</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Place and Route Completion</td>
<td>20</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Verify Timing Completion</td>
<td>20</td>
</tr>
<tr>
<td>Figure 29</td>
<td>FPGA Array Data Generated</td>
<td>20</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Design and Memory Initialization</td>
<td>21</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Fabric RAMs Tab</td>
<td>22</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Edit Fabric RAM Initialization Client Dialog Box</td>
<td>22</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Fabric RAM Content Applied</td>
<td>22</td>
</tr>
<tr>
<td>Figure 34</td>
<td>Generate Initialization Clients</td>
<td>23</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Generate Design Initialization Data Status</td>
<td>23</td>
</tr>
<tr>
<td>Figure 36</td>
<td>sNVM Client Verification</td>
<td>24</td>
</tr>
<tr>
<td>Figure 37</td>
<td>Generate Bitstream Completion</td>
<td>24</td>
</tr>
<tr>
<td>Figure 38</td>
<td>Board Setup</td>
<td>25</td>
</tr>
<tr>
<td>Figure 39</td>
<td>COM Port Number</td>
<td>26</td>
</tr>
<tr>
<td>Figure 40</td>
<td>Select Serial as the Connection Type</td>
<td>26</td>
</tr>
<tr>
<td>Figure 41</td>
<td>PuTTY Configuration</td>
<td>27</td>
</tr>
<tr>
<td>Figure 42</td>
<td>Run Program Action Completion</td>
<td>27</td>
</tr>
<tr>
<td>Figure 43</td>
<td>Hello World In Release Mode</td>
<td>28</td>
</tr>
<tr>
<td>Figure 44</td>
<td>Workspace Launcher</td>
<td>29</td>
</tr>
<tr>
<td>Figure 45</td>
<td>Creating New C Project</td>
<td>29</td>
</tr>
<tr>
<td>Figure 46</td>
<td>New Project Window</td>
<td>30</td>
</tr>
<tr>
<td>Figure 47</td>
<td>C Project Window</td>
<td>30</td>
</tr>
<tr>
<td>Figure 48</td>
<td>Empty Cortex-M1 Project</td>
<td>31</td>
</tr>
<tr>
<td>Figure 49</td>
<td>Firmware Catalog Window</td>
<td>31</td>
</tr>
<tr>
<td>Figure 50</td>
<td>Generate Options</td>
<td>32</td>
</tr>
<tr>
<td>Figure 51</td>
<td>CoreGPIO Files Report</td>
<td>32</td>
</tr>
<tr>
<td>Figure 52</td>
<td>CoreUARTapb Files Report</td>
<td>32</td>
</tr>
<tr>
<td>Figure 53</td>
<td>Import Option</td>
<td>33</td>
</tr>
<tr>
<td>Figure 54</td>
<td>Import Window</td>
<td>33</td>
</tr>
</tbody>
</table>
Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Tutorial Requirements</td>
<td>2</td>
</tr>
<tr>
<td>Table 2</td>
<td>Jumper Settings</td>
<td>25</td>
</tr>
</tbody>
</table>
1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the current publication.

1.1 Revision 7.0
Added Appendix 1: Running the TCL Script, page 51.

1.2 Revision 6.0
The following is a summary of the changes made in this revision.
• Updated the document for Libero SoC v12.2.
• Removed the references to Libero version numbers.

1.3 Revision 5.0
Updated the document for Libero® SoC v12.0.

1.4 Revision 4.0
Updated the document for Libero SoC PolarFire v2.2 release.

1.5 Revision 3.0
The following is a summary of the changes made in this revision.
• The document was updated for Libero SoC PolarFire v2.1.
• Replaced AND3 with CoreReset_PF throughout the document.
• Updated Connecting IP Blocks in SmartDesign, page 15.

1.6 Revision 2.0
The document was updated for Libero SoC PolarFire v2.0.

1.7 Revision 1.0
The first publication of this document.
2 Building a Cortex-M1 Processor Subsystem

Microsemi PolarFire® FPGAs support Cortex-M1 soft processors that can be used to run user applications. This tutorial explains how to build a Cortex-M1 processor subsystem using the Libero® SoC design suite. It lists the IP cores required to design a Cortex-M1 processor subsystem, describes how to configure and connect them and walks you through the Libero design flow to complete building it.

This tutorial also shows how to run the user application in release and debug mode on a PolarFire Evaluation Kit board. The application prints the string, Hello World! on the serial terminal, and blinks the LEDs on the board.

2.1 Requirements

The following table lists the hardware and software requirements for building the Cortex-M1 processor subsystem.

<table>
<thead>
<tr>
<th>Table 1 • Tutorial Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware</strong></td>
</tr>
<tr>
<td>Host PC</td>
</tr>
<tr>
<td>PolarFire Evaluation Kit (MPF300TS-EVAL-KIT)</td>
</tr>
<tr>
<td>— 12 V, 5 A AC power adapter and cord</td>
</tr>
<tr>
<td>— USB 2.0 A to Mini-B cable for UART and programming</td>
</tr>
<tr>
<td><strong>Software</strong></td>
</tr>
<tr>
<td>Libero SoC design suite</td>
</tr>
<tr>
<td>Firmware Catalog¹</td>
</tr>
<tr>
<td>SoftConsole</td>
</tr>
<tr>
<td>Serial Terminal Emulation Program</td>
</tr>
</tbody>
</table>

¹ Firmware catalog is included in the installation package of Libero SoC.

**Note:** Libero SmartDesign and configuration screen shots shown in this guide are for illustration purpose only. Open the Libero design to see the latest updates.
2.2 Prerequisites

Before you begin building a Cortex-M1 subsystem, all of the required components must be downloaded and installed as follows:

1. For demo design files download link: http://soc.microsemi.com/download/rsc/?f=mpf_tu0778_df
2. Download and install SoftConsole (as indicated in the website for this design) on the host PC from the following location: https://www.microsemi.com/products/fpga-soc/design-resources/design-software/softconsole#downloads
3. Download and install Libero SoC (as indicated in the website for this design) on the host PC from the following location: https://www.microsemi.com/product-directory/design-resources/1750-libero-soc#downloads
4. Start the Libero design suite application and download the latest versions of the following IP cores from Catalog:
   - CoreAHBtoAPB3: Bridge between the AHB and the APB domains.
   - CoreUARTapb: Controller for UART communication between the device and the host PC.
   - PF_SRAM_AHBL_AXI: Main memory of the Cortex-M1 soft processor.
   - CoreGPIO: Interface to enable the onboard LEDs.
   - CoreAHBLite: Bus interconnect for the AHB domain.
   - CoreAPB3: Bus to interface with the APB peripherals.
   - PF_INIT_MONITOR: Initialization monitoring resource to assert the device’s initialization.
   - PF_CCC: Clocking resource driving clocks to all the blocks in the design.
   - CORERESET_PF: Used to provide an asynchronous reset to all blocks in the design.
5. To download the licensed CORECORTEXM1 IP core:
   - Fill the Cortex-M1 agreement form available on the https://www.microsemi.com/form/91-coreip-cortex-m1 webpage.
   - Submit the form.

An email with the Cortex-M1 ZIP file is sent. Extract the ZIP file and import Cortex-M1.CPZ into the vault using the Add Core to Vault option as shown in the following figure.

```
Figure 1 • Add Core to Vault
```

You can now start building the Cortex-M1 processor subsystem in the Libero SoC.

2.3 Creating a Cortex-M1 Processor Subsystem

Creating a Cortex-M1 processor subsystem involves:

- Creating a Libero SoC Project, page 4
- Creating a New SmartDesign Component, page 7
- Instantiating the IP Cores in SmartDesign, page 7
- Connecting IP Blocks in SmartDesign, page 15
- Generating SmartDesign Component, page 17
- Managing Timing Constraints, page 18
- Running Libero Design Flow, page 18

The tutorial describes how to create an ARM Cortex-M1 subsystem for executing user applications. The user application can be stored in the sNVM, uPROM or SPI Flash. In this tutorial, the user application is stored in sNVM. At device power-up, the PolarFire System Controller initializes the designated LSRAMs with the user application from sNVM and releases the system reset. The Cortex-M1 soft processor exits the reset and starts executing the application. The user application prints the UART message “Hello World!” and blinks LEDs.
During the Libero design flow, the required non-volatile memory (sNVM, uPROM, or SPI Flash) must be specified for the fabric RAMs initialization. Then, the Fabric RAM initialization client must be created. The created fabric RAMs initialization client is stored in the sNVM, uPROM, or SPI Flash according to the user selection.

The following figure shows the top-level block diagram of the design.

**Figure 2 • Block Diagram**

This section describes how to perform all the procedures required to create a Cortex-M1 processor subsystem in a new SmartDesign canvas.

### 2.3.1 Creating a Libero SoC Project

To create a Libero SoC project, perform the following steps:

1. From the Libero SoC Menu bar, click **Project > New Project**.
2. Enter the following **New Project** information as shown in the following figure and click **Next**.
   - **Project name**: PolarFire_CortexM1_Subsystem
   - **Project location**: Select an appropriate location (for example, F:/Microsemi/CortexM1_Project)
   - **Preferred HDL type**: Verilog
Figure 3 • New Project Details

3. Select the following values using the drop-down list for Device Selection as shown in the following figure and click Next.
   • Family: PolarFire
   • Die: MPF300T
   • Package: FCG1152
   • Speed: STD
   • Range: EXT
   • Part Number: MPF300T-1FCG1152E

Figure 4 • Device Selection
4. To retain the default Core Voltage and I/O settings, click Next in the Device Settings window.

**Figure 5 • Device Settings**

5. In the Add HDL Sources window, click Next to go to the next step because HDL files are unused.

6. In the Add constraints window, click Import file to import the I/O constraint file as shown in the following figure.

**Figure 6 • Add Constraints Window**

7. In the Import files window, locate the user_io.pdc file in the DesignFiles_directory\Source folder, and double-click it.

8. Click Finish.

   The following message displayed in the Log pane:

   The PolarFire_CortexM1_Subsystem project was created.

   The Reports tab is highlighted and the project details are printed to the PolarFire_CortexM1_Subsystem.log file.

   The Libero SoC project for PolarFire Cortex-M1 design is successfully created.
2.3.2 Creating a New SmartDesign Component

To create a new SmartDesign component, perform the following steps:

1. Select File > New > SmartDesign.
2. In the Create New SmartDesign dialog box, enter CortexM1_Subsystem as the name of the new SmartDesign project, as shown in the following figure.
3. Click OK.

Figure 7 • Create New SmartDesign

The CortexM1_Subsystem SmartDesign tab opens. The CortexM1_Subsystem SmartDesign component is successfully created. Next, we need to instantiate, configure, and connect the IP Cores required to build the processor subsystem.

2.3.3 Instantiating the IP Cores in SmartDesign

After an IP core is dragged into SmartDesign, Libero displays the Create Component window. A component name for the IP core must be entered in this window. After naming the component, the configurator of that IP core is displayed and after configuring the IP core, Libero generates the design component of that IP core and instantiates it in SmartDesign. HDL files can be dragged into SmartDesign and instantiated directly. For the recent version of IP Cores, refer Table 1, page 2.

2.3.3.1 Instantiating CORERESET_PF

Note: The version used in this design is 2.1.100.

To instantiate and configure CORERESET_PF:

1. From the Catalog, find and drag CORERESET_PF IP into SmartDesign.
2. In the Create Component window, enter pf_reset as the component name.
3. In the CoreReset_PF configurator, retain the default configuration and click OK.
4. The CoreReset_PF IP is successfully instantiated in SmartDesign.

2.3.3.2 Instantiating PF_INIT_MONITOR

To instantiate PF_INIT_MONITOR, perform the following steps:

1. From theCatalog, find and drag the PolarFire Initialization Monitor IP core into SmartDesign.
2. In the Create Component window, enter PF_INIT_MONITOR_0 as the Component name and click OK.
3. In the PF_INIT_MONITOR Configurator, Uncheck Enable Bank0, Enable Bank1, Enable Bank2, and Enable Bank4 calibration status pins as shown in Figure 8, page 8 and click OK.

The PF_INIT_MONITOR IP component is successfully instantiated and generated.
2.3.3.3 Instantiating PF_CCC

To instantiate PF_CCC, perform the following steps:

1. From the Catalog, find and drag the PF_CCC IP core into SmartDesign.
2. In the Create Component window, enter PF_CCC_0 as the Component name and click OK.
3. In the PF_CCC Configurator:
   - Retain the configuration to PLL-Single.
   - In the Clock Options PLL tab (Figure 9, page 9), set the Input Frequency to 50 MHz and Bandwidth to High.
   - Set the Power/Jitter to Maximize VCO for Lowest Jitter.
   - Set the Feedback Mode to Post-VCO.
   - In the Output Clocks tab > Output Clock 0 pane (Figure 10, page 9), ensure that the Enabled checkbox is selected.
   - Requested Frequency is set to 80 MHz. Ensure that the Global Clock checkbox is selected.
   - Click OK.
There is a possibility of a warning message to check the log window. Click OK to proceed further.
The PF_CCC IP component is successfully instantiated and generated.
2.3.3.4 Instantiating CoreCORTEXM1

To instantiate CoreCORTEXM1, perform the following steps:

1. From the Catalog, find and drag the CoreCORTEXM1 into SmartDesign.
2. In the Create Component dialog box, enter CoretxM1_0 as the component name and click OK.
3. In the CoreCORTEXM1 Configurator, set the Debug Interface to FlashPro and ensure that the Include reset control logic check box is selected, as shown in the following figure.
4. Click OK.

*Figure 11 • CoreCORTEXM1 Configurator*

The CoreCORTEXM1 IP component is successfully instantiated and generated.
2.3.3.5 Instantiating CoreAHBLite

To instantiate CoreAHBLite, perform the following steps:

1. From the Catalog, find and drag the CoreAHBLite IP core into SmartDesign.
2. In the Create Component window, enter coreabhlite_0 as the component name and click OK.
3. In the CoreAHBLite Configurator, do the following settings as shown in the following figure:
   • Set the Memory space to 1 MB addressable space apportioned into 16 slave slots, each of size 64 KB.
   • From the Enable Master Access pane select only the M0 can access slot 0 and M0 can access slot 4.
   • Click OK.

Figure 12 • CoreAHBLite Configurator

The CoreAHBLite IP component is successfully instantiated and generated.

2.3.3.6 Instantiating PF_SRAM_AHBL_AXI

To instantiate PF_SRAM_AHBL_AXI, perform the following steps:

1. From the Catalog, find and drag the PF_SRAM_AHBL_AXI IP core into SmartDesign.
2. In the Create Component window, enter PF_SRAM as the component name and click OK.
3. In the PF_SRAM_AHBL_AXI Configurator, do the following settings as shown in Figure 13, page 12:
   • Set the SRAM type to LSRAM.
   • Set the Memory Depth to 16384 to create 64 KB (16384 × 4 bytes) memory.
   • Set the Fabric Interface type to AHBLite.
   • Click Finish.
2.3.3.7 Instantiating CoreAHBtoAPB3

To instantiate CoreAHBtoAPB3, perform the following steps:

1. From the Catalog, find and drag the CoreAHBtoAPB3 IP core into SmartDesign.
2. In the Create Component window, enter core_ahb_to_apb3 as the component name and click OK.
3. In the Configurator, retain the default configuration settings and click OK.

The COREAHBTOAPB3 IP component is successfully instantiated and generated.

2.3.3.8 Instantiating CoreAPB3

To instantiate CoreAPB3, perform the following steps:

1. From the Catalog, find and drag the CoreAPB3 IP core into SmartDesign.
2. In the Create Component window, enter CoreAPB3_0 as the component name and click OK.
3. In the CoreAPB3 Configurator:
   • Select the Data Width Configuration pane.
   • In the Data Width Configuration pane, retain the APB Master Data Bus Width value as 32-bit.
   • In the Address Configuration pane, set Number of address bits driven by master to 16.
   • Set Position in slave address of upper 4 bits of master address to [27:24] (This value is not entered if master address width >= 32 bits).
   • In the Enabled APB Slave Slots pane, select Slot 0 and Slot 1. Clear all the other slots.
   • Click OK.
2.3.3.9 Instantiating CoreGPIO

To instantiate CoreGPIO, perform the following steps:

1. From the Catalog, find and drag the CoreGPIO IP core into SmartDesign.
2. In the Create Component window, enter CoreGPIO_0 as the component name and click OK.
3. In the CoreGPIO Configurator:
   - Select the Global Configuration pane.
   - In the Global Configuration pane, set APB Data Width to 32 and Output enable to Internal.
   - Set Number of I/Os to 4.
   - Set Single-bit interrupt port to Disabled.
   - In the I/O bit 0, I/O bit 1, I/O bit 2, and I/O bit 3 panes, select Fixed Config.
   - Set I/O Type to Output.
   - Set the Interrupt Type to Disabled.
   - Click OK.
2.3.3.10 Instantiating CoreUARTapb

To instantiate CoreUARTapb, perform the following steps:

1. From the Catalog, find and drag the CoreUARTapb IP core into SmartDesign.
2. In the Create Component window, enter CoreUARTapb_0 as the component name and click OK.
3. In the CoreUARTapb Configurator, retain the default configuration settings and click OK.

The CoreUARTapb IP component is successfully instantiated and generated.

The following figure shows the CortexM1_Subsystem in SmartDesign after instantiating and configuring the IP blocks.

Figure 16 • CortexM1_Subsystem Without Connections
2.3.4 Connecting IP Blocks in SmartDesign

Connect IP blocks in CortexM1_Subsystem using any of the following connection methods:

• **Using the Connection Mode option:** In this method, change the SmartDesign to Connection Mode by clicking **Connection Mode** on the SmartDesign window, as shown in the following figure. The cursor changes from the normal arrow shape to the connection mode icon shape. To make a connection in this mode, click on the first pin and drag-drop to the second pin that you want to connect.

![Connection Method](image)

• The other method is by selecting the pins to be connected together and selecting **Connect** from the context menu. To select multiple pins to be connected together, press down the Ctrl key while selecting the pins. Right-click the input source signal and select **Connect** to connect all the signals together. Similarly, select the input source signal, right-click it, and select **Disconnect** to disconnect the already connected signals.

Using any of the preceding methods, make the following connections:

1. Perform the following pin settings on **PF_INIT_MONITOR_0**:
   - Select **FABRIC_POR_N, PCIE_INIT_DONE, USRAM_INIT_DONE, SRAM_INIT_DONE, XCVR_INIT_DONE, USRAM_INIT_FROM_SNVM_DONE, USRAM_INIT_FROM_UPROM_DONE, USRAM_INIT_FROM_SPI_DONE, SRAM_INIT_FROM_SNVM_DONE, SRAM_INIT_FROM_UPROM_DONE, SRAM_INIT_FROM_SPI_DONE, AUTOCALIB_DONE**, and right-click all of these pins and select **Mark Unused**.
   - Connect the **DEVICE_INIT_DONE** pin to **pf_reset: INIT_DONE** pin.

2. Perform the following pin settings on **pf_reset**:
   - Right-click **EXT_RST_N**, select **Promote to Top Level**, and then rename it to **RESETN**.
   - Connect **FABRIC_RESET_N** to **CoretxM1_0: SYSRESETN, DBGRESETN, and NRESET** pins.
   - Right-click **SS_BUSY** and **FF_US_RESTORE** and select **Tie Low**.

3. Perform the following pin settings on **PF_CCC_0**:
   - Right-click the **REF_CLK_0** pin and select **Promote to Top Level**.
   - Connect the **PLL_LOCK_0** pin to **pf_reset: PLL_LOCK**.
   - Connect the **OUT0_FABCLK_0** pin to the following listed pins:
     - **pf_reset: CLK**
     - **CoretxM1_0: HCLK**
     - **PF_SRAM: HCLK**
     - **core_ahb_to_apb3: HCLK**
     - **coreahlite_0: HCLK**
     - **CoreUARTapb_0: PCLK**
     - **CoreGPIO_0: PCLK**

4. Perform the following pin settings on **CoretxM1_0**:
   - Right-click **WDGRES, NMI, EDBGQR, DBGRESTART, IRQ0, and IRQ1 to 31** pins and select **Tie Low**.
   - Select the **SWCLKTCK, NTRST, SWDITMS, TDI, and TDO** pins. Right-click and select **Promote to Top Level**.
   - Connect the **AHB_MASTER** pin to **coreahlite_0: AHBmmaster0 (mirroredMaster)**.
   - Right-click the **TDO** pin and select **Promote to Top Level**.
   - Right-click **WDGRESN, LOCKUP, HALTED, SYSRESETREQ, JTAGTOP, JTAGNSW, DBGRESTARTED** and select **Mark Unused**.
• Connect the HRESETN pin as shown in the following list:
  • coreahblite_0: HRESETN
  • PF_SRAM: HRESETN
  • core_ahb_to_apb3: HRESETN
  • CoreUARTapb_0: PRESETN
  • CoreGPIO_0: PRESETN
5. Connect coreahblite_0: AHBmslave0 (mirroredSlave) to PF_SRAM: AHBSlaveInterface.
6. Connect coreahblite_0: AHBmslave4 (mirroredSlave) to core_ahb_to_apb3: AHBSlave.
7. Connect core_ahb_to_apb3: APBmaster to APB3_0: APB3master (mirroredMaster).
8. Right-click the REMAP_M0 and select Tie Low.
9. Connect CoreAPB3_0: APBslave0 to CoreGPIO_0: APB_bif and CoreAPB3_0: APBslave1 to CoreUARTapb_0: APB_bif
10. Perform the following pin settings on CoreUARTapb_0:
    • Right-click the RX Pin and select Promote to Top Level.
    • Right-click TXRDY, RXRDY, PARITY_ERR, OVERFLOW, and FRAMING_ERR pins and select Mark Unused.
    • Right-click the TX pin and select Promote to Top Level.
11. Perform the following pin settings on CoreGPIO_0:
    • Right-click the GPIO_IN [3:0] pin and select Tie Low.
    • Right-click the INT [3:0] pin and select Mark Unused.
    • Right-click the GPIO_OUT [3:0] pin and select Promote to Top Level.
12. Click File > Save CortexM1Subsystem.

The IP blocks are successfully connected.

The following figure shows the CortexM1Subsystem in SmartDesign after connecting all IP blocks.

Figure 18 • CortexM1Subsystem With Connections

The Cortex-M1 processor subsystem is successfully designed in SmartDesign. The system address map can be viewed by right-clicking the SmartDesign canvas and selecting the Modify Memory Map option. The Modify Memory Map dialog box is shown in the following figure and Figure 20, page 17 for APB3 and AHBLite peripherals.
Now generate the SmartDesign component and run the Libero design flow.

2.3.5 Generating SmartDesign Component

To generate the component, perform the following steps:

1. In Design Hierarchy, click the Build Hierarchy option as shown in the following figure.

2. Save the project.

3. Click Generate Component button on the SmartDesign toolbar. The following figure shows the Generate Component button.
2.3.6 Managing Timing Constraints

Before running the Libero design flow, derive timing constraints as explained in the following sections.

2.3.6.1 Deriving Constraints

Derive the timing constraints using the Derived Constraints option available in the Timing tab of the Manage Constraints window.

To derive constraints, perform the following steps:

2. In the Manage Constraints window, select the Timing tab, and click Derive Constraints.
   - The design hierarchy is built again. In the Message alert box, click Yes to attach the derived constraints SDC file to the Synthesis, Place and Route, and Timing Verification.
3. The CortexM1_Subsystem_derived_constraints.sdc file is generated in the project folder. Click Yes in the alert box to associate the derived constraint SDC file to the Synthesis, Place and Route, and Timing Verification tools as shown in the following figure.

The derived constraints SDC file is generated successfully. After including the timing constraint files, the design flow described in the following sections must be executed to build Cortex-M1 processor subsystem on the PolarFire device.

2.3.7 Running Libero Design Flow

The Libero design flow involves running the following processes:

- Synthesis, page 19
- Place and Route, page 19
- Verify Timing, page 20
- Generate FPGA Array Data, page 20
- Configure Design Initialization Data and Memories, page 21
- Generate Bitstream, page 24
- Run Program Action, page 25
2.3.7.1 Synthesis
To synthesize the design, perform the following steps:

1. Double-click Synthesis from the Design Flow window to synthesize the design component. A green tick mark is displayed after the successful completion of the synthesis process as shown in the following figure.

   Figure 24 • Synthesis Completion

2. On the Reports window, see the Synthesis report and log files.

2.3.7.2 Place and Route
The Place and Route process requires I/O. The I/O constraints file user_io.pdc was imported while creating the libero project. The user_io.pdc file must be mapped. This file is available in the design files folder at DesignFiles_Directory\Source folder.

To map the I/O constraints, perform the following steps:

1. Double-click Manage Constraints from the Design Flow window as shown in the following figure.

   Figure 25 • Manage Constraints

2. In the Manage Constraints window, select the I/O Attributes tab and select the check box next to the user_io.pdc file as shown in the following figure.

   Figure 26 • I/O Attributes

3. Save the project.
The I/O constraint file is successfully mapped. Now, double-click Place and Route from the Design Flow window.

A green tick mark is displayed after the successful completion of the Place and Route process, as shown in the following figure.
In the **Reports** window, see the Place and Route report and log files.

### 2.3.7.3 Verify Timing
To verify timing, perform the following steps:

1. On the **Design Flow** window, double-click **Verify Timing**.  
   A green tick mark is displayed after the successful completion of the verify timing process as shown in the following figure.

### 2.3.7.4 Generate FPGA Array Data
To generate FPGA array data, perform the following step:

On the **Design Flow** window, double-click **Generate FPGA Array Data**.  
A green tick mark is displayed after the successful generation of the FPGA array data as shown in the following figure.
2.3.7.5 Configure Design Initialization Data and Memories

This process requires the user application executable file (hex file) as input to initialize the LSRAM blocks after device power-up. The hex file is provided along with the design files. For more information about building the user application, see Creating User Application Using SoftConsole, page 29.

The hex file (m1fpga-cortex-m1-blinky.hex) is available in the DesignFiles_Directory\Source folder. When the hex file is imported, a memory initialization client is generated for LSRAM blocks. If the SoftConsole project is regenerated, ensure to delete the first line in the .hex file. The type of .hex file used here is Release Mode Generated .hex file. The first line is deleted in the .hex file provided with the design files.

**Note:** To make the .hex file generated by SoftConsole compatible with the process of configuring design initialization data and memories in Libero, delete the extended linear record present in the first line of the .hex file. The .hex file available in the DesignFiles_Directory\Source folder is already modified to be compatible.

To create the memory initialization client, perform the following steps:

1. On the Design Flow window, double-click Configure Design Initialization Data and Memories. The Design and Memory Initialization window opens as shown in the following figure.

**Figure 30 • Design and Memory Initialization**

![Diagram showing Design and Memory Initialization process](image)

2. Select the Fabric RAMs tab and select the CortexM1_Subsystem/PF_SRAM client from the list and click Edit as shown in the following figure.
3. In the Edit Fabric RAM Initialization Client dialog box, select the Content from file option, locate the mlfpga-cortex-m1-blinky.hex file from DesignFiles_directory\Source folder and Click OK as shown in the following figure.

4. Click Apply as shown in the following figure.
5. Select the Design Initialization tab and configure the following option, Memory type for third stage initialization client: sNVM, as shown in following figure.

**Figure 34 • Generate Initialization Clients**

![Design Initialization tab configuration](image)

6. Double-click Generate Design Initialization Data option in the Libero design flow to generate the initialization clients in sNVM memory.

7. When the initialization clients are generated, the status is displayed in the Log window as shown in the following figure.

**Figure 35 • Generate Design Initialization Data Status**

![Log window with initialization status](image)
8. Select the sNVM tab to verify that the SNVM client is generated as shown in Figure 36, page 24. The INIT_STAGE_2_3_SNVM_CLIENT indicates that the sNVM client was successfully generated.

![Figure 36 • sNVM Client Verification](image)

The process of configuring design initialization data and memories is successfully completed.

2.3.7.6 Generate Bitstream

To generate bitstream, perform the following step:

On the Design Flow window, double-click Generate Bitstream.

A green tick mark is displayed after the successful generation of the bitstream as shown in the following figure.

![Figure 37 • Generate Bitstream Completion](image)

On the Reports window, see the corresponding log files.
2.3.7.7 Run Program Action

After generating the bitstream, set up the PolarFire Evaluation Kit board so that the device is ready to be programmed. Also, set up the serial terminal emulation program (PuTTY) to observe the output of the user application.

Figure 38 • Board Setup

2.3.7.7.1 Board Setup

To set up the board, perform the following steps:

1. Ensure that the jumper settings on the board are same as listed in the following table.

Table 2 • Jumper Settings

<table>
<thead>
<tr>
<th>Jumper</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>J18, J19, J20, J21, and J22</td>
<td>Close pin 2 and 3 for programming the PolarFire FPGA through FTDI</td>
</tr>
<tr>
<td>J28</td>
<td>Close pin 1 and 2 for programming through the on-board FlashPro5</td>
</tr>
<tr>
<td>J4</td>
<td>Close pin 1 and 2 for manual power switching using SW3</td>
</tr>
<tr>
<td>J12</td>
<td>Close pin 3 and 4 for 2.5 V</td>
</tr>
</tbody>
</table>

2. Connect the power supply cable to the J9 connector on the board.
3. Connect the USB cable from the Host PC to the J5 (FTDI port) on the board.
4. Power on the board using the SW3 slide switch.

The board is successfully set up.
2.3.7.7.2 Serial Terminal Emulation Program (PuTTY) Setup

The user application prints the string, “Hello World!” on the serial terminal through the UART interface.

To setup the serial terminal program, perform the following steps:

1. Ensure that the USB cable is connected from the host PC to the J5 (USB) port on the PolarFire Evaluation Kit board.
2. Start the PuTTY program.
3. Start Device Manager, note the second highest COM Port number and use that in the PuTTY configuration. For example, COM Port 93 is used in this instance as shown in the following figure. COM Port numbers may vary.

   **Figure 39 • COM Port Number**

   ![COM Port Number](image)

4. Select Serial as the Connection type as shown in the following figure.

   **Figure 40 • Select Serial as the Connection Type**

   ![Select Serial as the Connection Type](image)

5. Set the Serial line to connect to COM port number noted in Step 3.
6. Set the Speed (baud) to 115200 as shown in the following figure.
7. Set the Flow control to None as shown in the following figure and click Open.
PuTTY opens successfully, and this completes the serial terminal emulation program setup.

To program the PolarFire device, double-click **Run PROGRAM Action** from the **Libero > Design Flow** tab. A green tick mark is displayed after the successful completion of the Run Program Action process as shown in the following figure.

When the device is successfully programmed, the device gets reset and performs the following sequence of operations:
1. The PolarFire System Controller initializes the LSRAM with the user application code from sNVM and releases the system reset.
2. The CORTEX-M1 processor completes the reset and executes the user application from LSRAM. As a result, LEDs 4, 5, 6, and 7 blink and the string, "Hello World!" is printed on the PuTTY as shown in the following figure.

**Figure 43 • Hello World In Release Mode**

The Cortex-M1 processor subsystem is successfully built and programmed on the board.
This section describes how to create and debug the Cortex-M1 application using SoftConsole.

Creating the user application involves:

- Creating a Cortex-M1 Project, page 29
- Downloading the Firmware Drivers, page 31
- Importing the Firmware Drivers, page 33
- Creating the main.c File, page 34
- Configuring the Cortex-M1 Project, page 35
- Mapping Memory and Peripheral Addresses, page 40
- Setting the UART Baud Rate, page 41
- Building the User Application in Release Mode, page 42
- Building In Debug Mode and Debugging the User Application, page 44

3.1 Creating a Cortex-M1 Project

To create a Cortex-M1 project, perform the following steps:

1. Create a SoftConsole workspace folder on the host PC for storing SoftConsole projects. For example, F:\Tutorial\CortexM1.
2. Start SoftConsole. In the Workspace Launcher dialog box, paste F:\Tutorial\CortexM1 as the workspace location and click OK as shown in the following figure.

![Workspace Launcher](image)

When the workspace is successfully launched, the metadata and the RemoteSystemsTempFiles folders are created in the workspace directory. The SoftConsole main window opens.

3. Select File > New > Project as shown in the following figure.

![Creating New C Project](image)
4. In the **New Project** window, expand **C/C++**, select **C Project**, and then, select **Next**.

*Figure 46 • New Project Window*

![](image1)

5. In the **Project type** pane:
   - Enter a name for the project in the **Project name** field. For example, `m1fpga-cortex-m1-blinky`.
   - Expand **Executable** and select **Empty Project** as shown in the following figure and then, click **Next**.

*Figure 47 • C Project Window*

6. In the **Select Configurations** window, select **Debug** and **Release**, and then click **Next**.

7. Retain the default Toolchain name and Toolchain path, and then click **Finish**.
An empty Cortex-M1 project (mlfpga-cortex-m1-blinky) is created in Debug and Release mode as shown in the following figure.

Figure 48 • Empty Cortex-M1 Project

The Cortex-M1 project is successfully created.

3.2 Downloading the Firmware Drivers

The empty Cortex-M1 project requires the hardware abstraction layer (HAL) files, Cortex microcontroller software interface standard (CMSIS) files, and the following peripheral drivers:

- CoreGPIO
- CoreUARTapb

Download the peripheral drivers using the Firmware Catalog application. This application is installed during Libero installation. For more information on Firmware Catalog, refer Prerequisites, page 3.

To download the drivers, perform the following steps:

1. Create a folder named firmware in the CortexM1 project workspace.
2. Start Firmware Catalog. The following figure shows the Firmware Catalog window.

Figure 49 • Firmware Catalog Window

3. If new cores are available, click Download them now!
4. In the **Firmware Catalog** window, right-click the latest **CoreGPIO Driver**, and select **Generate**.

5. In the **Generate Options** window, locate the folder named **firmware** and click **OK**.

   ![Generate Options](image1)

   When the files are generated, the **Reports** window lists the files generated as shown in the following figure.

   ![CoreGPIO Files Report](image2)

6. In the **Firmware Catalog** window, right-click the latest **CoreUARTapb Driver** and select **Generate**.

7. In the **Generate Options** window, enter `F:\Tutorial\CortexM1\firmware` as the **Project folder**, and click **OK**.

   When the files are generated, the **Reports** window lists the files generated as shown in the following figure.

   ![CoreUARTapb Files Report](image3)

8. Copy the following folders and files from `DesignFiles_Directory\Source` to `F:\Tutorial\CortexM1\firmware` at the **Project folder**.
   - CMSIS
   - hal
   - blinky.ld
   - hw_platform.h

   This completes the copying of CMSIS and HALs files requirements.

   HAL files and firmware drivers are downloaded.
3.3 Importing the Firmware Drivers

After downloading the drivers, CMSIS, and HAL files, import them into the empty Cortex-M1 project created.

To import the drivers, perform the following steps:

1. In SoftConsole, right-click the `m1fpga-cortex-m1-blinky` project, and select `Import` as shown in the following figure.

   *Figure 53 - Import Option*

2. In the `Import` window, expand the `General` folder and double-click `File System` as shown in the following figure.

   *Figure 54 - Import Window*

3. In the continued `Import` window, do the following steps (see Figure 55, page 34):
   - Click `Browse` and locate the `F:\Tutorial\CortexM1\firmware` folder
   - Select the `firmware` folder and click `OK`.
   - Expand the `firmware` folder and select all the checkbox as shown in the following figure.
   - Click `Finish`.
Creating User Application Using SoftConsole

3.4 Creating the main.c File

To update the main.c file, perform the following steps:

1. In the Menu bar, click File and select New > Source File.
2. In the New Source File dialog box, enter main.c in the Source file field and click Finish as shown in the following figure.

The main.c file is created inside the project as shown in the following figure.
3. Copy all of the content of the DesignFiles_directory\Source\main.c file and paste it in the main.c file of the SoftConsole project.
This updates the main.c file.

3.5 Configuring the Cortex-M1 Project

At this stage, the location of CMSIS, drivers and HAL files are not mapped.

To map CMSIS, drivers and HAL files, perform the following steps:
1. In the project explorer, right-click the mlfpga-cortex-m1-blinky project and select Properties.
2. Expand C/C++ Build and select Settings.
3. Set the Configuration to All Configurations as shown in the following figure.

Figure 58 • All Configuration Setting

4. In the Tool Settings, expand Target Processor, and set the ARM family to cortex-m1.
5. Retain all the other default settings, as shown in the following figure:
6. In the Tool Settings tab, expand GNU ARM CROSS C Compiler and select Includes.
7. To add driver, HAL, and CMSIS directory paths, click Add as shown in the following figure.

![Figure 59 • Target Processor](image)

![Figure 60 • Tool Settings Options](image)
8. In the **Add directory path** dialog box, click **Workspace** as shown in the following figure.

*Figure 61 • Adding CoreGPIO Directory Path*

9. In the **Folder Selection** dialog box, expand **mifpga-cortex-m1-blinky project > drivers** and select the **CoreGPIO** and **CoreUARTapb** folders and click **OK**, as shown in the following figure.

*Figure 62 • Adding the CoreGPIO Folder*

10. In the **Add directory path** dialog box, click **OK**.
    The **CoreGPIO** and the **CoreUARTapb** folder paths are added as shown in the following figure.

*Figure 63 • CoreGPIO Path Added*
11. Similarly, add the other paths as shown in the following figures.

*Figure 64 • Adding CMSIS and startup_gcc paths*

*Figure 65 • Adding HAL, CortexM1, and GNU Paths*

12. In **GNU ARM CROSS C Compiler**, select **Miscellaneous** and set the **Other compiler flags** to:

```
--specs=cmsis.specs
```

(as shown in the following figure).

*Figure 66 • Miscellaneous Setting*

13. Click **OK**.
The CMSIS, drivers, and HAL directory paths are successfully mapped as shown in the following figure.

**Figure 67 • Mapping Successful**

14. From GNU ARM CROSS C Linker > General use the Add option to map the `blinky.ld` linker script.

15. In the Add file path window, click Workspace, and expand the `m1fpga-cortex-m1-blinky` project and select the `blinky.ld` file, as shown in the following figure.

**Figure 68 • Mapping Linker Script**

16. Click **OK**.
3.6 Mapping Memory and Peripheral Addresses

In the Libero design flow, the Cortex-M1 processor execution memory address is mapped to 0x00000000 and its size is set to 24 KB. This information must be provided in the linker script for building the application.

To map the memory address, perform the following steps:

1. Open the linker script (`blinky.ld`) available in the `m1fpga-cortex-m1-blinky` folder.
2. Ensure that the ram ORIGIN address is mapped to 0x00000000.
3. Change the LENGTH of the ram to 24 KB.
4. Ensure that the RAM_START_ADDRESS is mapped to 0x00000000.
5. Change the RAM_SIZE to 24 KB.
6. Change the STACK_SIZE to 4 KB.
7. Change the HEAP_SIZE to 4 KB.
8. Save the file.

The following figure shows the linker script after these updates.

![Updated Linker Script](image)

In the Libero design flow, the addresses of GPIO and UART peripherals are mapped to 0x00040000 and 0x00041000 respectively. This information needs to be provided in the `main.c` file provided in `m1fpga-cortex-m1-blinky` project folder.
To map the peripheral address, perform the following steps:

1. Open the main.c file.
2. Look for the COREGPIO_BASE_ADDR macro and define it as 0x00040000UL.
3. Look for the UART_BASE_ADDRESS macro and define it as 0x00041000UL.
4. Save the file.

The following figure shows the main.c after these updates.

![Updated main.c File](image)

The memory and peripheral addresses are successfully mapped.

### 3.7 Setting the UART Baud Rate

The value of the BAUD_VALUE macro in the main.c file must be defined according to the system clock frequency to achieve the UART baud rate of 115200. The baud value is calculated by the UART_init function in the main.c file.

To define the baud value, perform the following steps:

1. Look for the #define BAUD_VALUE statement in the main.c file.
2. Define it as:
   ```c
   #define BAUD_VALUE 115200
   ```

![Defining Baud Value](image)

The UART baud rate is successfully set.
### 3.8 Building the User Application in Release Mode

To build the user application in the release mode, perform the following steps:

1. Right-click the **project** and select **Build Configurations > Set Active > Release** as shown in the following figure.

   ![Build Configuration](image1)

   **Figure 73 • Build Configuration**

2. Right-click the **project** and select **Build Project** as shown in the following figure.

   ![Build Project](image2)

   **Figure 74 • Build Project**
3. SoftConsole builds the project and displays “Build Finished” message in the log window, as shown in the following figure.

*Figure 75 • Build Finished*

4. The user application file (.hex) is generated in the **Release** folder as shown in the following figure.

*Figure 76 • HEX File*

This file must be imported to Libero for generating the initialization client and for adding the client to sNVM for initializing the SRAM block at device power-up.

**Note:** The .hex file generated here is used in Configure Design Initialization Data and Memories, page 21.
3.9 Building In Debug Mode and Debugging the User Application

To build the application, perform the following steps:

1. Right-click the project and select Build Configurations > Set Active > Debug as shown in the following figure.

2. On the Project Explorer, right-click the mlfpga-cortex-ml-blinky project and select Build Project.
3. Ensure that no errors are displayed in the build result. Before debugging, the board and the serial terminal must be set up.

For more information about the board and serial terminal setup, see Board Setup, page 25 and Serial Terminal Emulation Program (PuTTY) Setup, page 26.

To debug the application, perform the following steps:

1. On the Project Explorer, select the mlfpga-cortex-m1-blinky project.
2. Select Run > Debug Configurations from the SoftConsole toolbar as shown in the following figure.
3. In the Debug Configurations dialog box, do the following steps (see the following figure):
   - To generate debug configuration for mlfpga-cortex-m1-blinky project, double-click GDB OpenOCD Debugging.
   - Select the generated mlfpga-cortex-m1-blinky Debug configuration and select Search Project, as shown in the following figure.
4. In the **Program Selection** window, select the **elf** file as shown in the following figure, and click **OK**.

*Figure 81 • Program Selection*

![Program Selection](image)

5. In the **Debugger** tab, replace the **Config Options**, **Executable**, and **Commands** entries with the following entries as shown in the following figure.
   - **Config Options**: --file board/microsemi-cortex-m1.cfg
   - **Executable**: ${cross_prefix}gdb${cross_suffix}
   - **Commands**: set mem inaccessible-by-default off

*Figure 82 • Settings in the Debugger Tab*

![Settings in the Debugger Tab](image)

6. In **Debug Configurations -> Startup** tab, clear the Pre-run/Restart reset check box to halt the program at the **main ()** function and clear the Enable ARM semihosting checkbox.
7. Click **Debug**.
8. The **Confirm Perspective Switch** dialog box opens as shown in the following figure.

**Figure 83** • **Debug Settings- Startup Tab**

The debugger copies the executable code to LSRAM and halts the execution at the first instruction in the `main.c` file as shown in the following figure. The Cortex-M1 processor executes the code from LSRAM.

9. Click **Yes**.

The debugger copies the executable code to LSRAM and halts the execution at the first instruction in the `main.c` file as shown in the following figure. The Cortex-M1 processor executes the code from LSRAM.
10. To resume the application execution, click **Resume** on the **SoftConsole** toolbar, as shown in the following figure.

**Figure 86 • Resume Application Execution**
11. The string, "Hello World!" is printed on the serial terminal as shown in the following figure. Also, LEDs 4, 5, 6, and 7 blink on the PolarFire Evaluation Kit board.

*Figure 87 • Hello World in Debug Mode*

12. To suspend the application execution, select **Run > Suspend** in the SoftConsole Menu bar.

13. To view the values of the Cortex-M1 internal registers, click **Registers**, as shown in the following figure.

*Figure 88 • Cortex-M1 Register Values*

14. To view the values of variables in the source code, click **Variables**.

15. On the SoftConsole toolbar, use the **Step Over** option to view the application execution line by line, or use the **Step Into** option to execute the instructions inside a function. Use the **Step Return** option to come out the function. You can also add breakpoints in the application source code.

16. To terminate the debugging of the application, click **Terminate** on the **SoftConsole** toolbar.


This concludes the debugging process of this tutorial.
4 Appendix 1: Running the TCL Script

TCL scripts are provided in the design files folder under directory TCL_Scripts. If required, the design flow can be reproduced from Design Implementation till generation of job file.

To run the TCL, follow the steps below:

1. Launch the Libero software
2. Select Project > Execute Script....
3. Click Browse and select script.tcl from the downloaded TCL_Scripts directory.
4. Click Run.

After successful execution of TCL script, Libero project is created within TCL_Scripts directory.

For more information about TCL scripts, refer to mpf_tu0778_df/TCL_Scripts/readme.txt.

Refer to Libero® SoC TCL Command Reference Guide for more details on TCL commands. Contact Technical Support for any queries encountered when running the TCL script.
Appendix 2: References

This section lists the documents that provide more information about Cortex-M1 and other IP cores used to build the Cortex-M1 subsystem.

- For more information about the CORECORTEXM1 IP core, see CoreCortexM1_HB.pdf from Libero->Catalog.
- For more information about the CoreAHBtoAPB3 IP core, see CoreAHBtoAPB3_HB.pdf.
- For more information about the CoreGPIO IP core, see CoreGPIO_HB.pdf.
- For more information about the CoreUARTapb IP core, see CoreUARTapb_HB.pdf.
- For more information about the CoreAHBLite IP core, see CoreAHBLite_HB.pdf.
- For more information about the CoreAPB3 IP core, see CoreAPB3_HB.pdf.
- For more information about the PF_INIT_MONITOR IP core, see UG0725: PolarFire FPGA Device Power-Up and Resets User Guide.
- For more information about the PF_CCC IP core, see UG0684: PolarFire FPGA Clocking Resources User Guide.
- For more information about the PF_SRAM_AHBL_AXI IP core, see UG0680: PolarFire FPGA Fabric User Guide.
- For more information about Libero, ModelSim, and Synplify, see Microsemi Libero SoC webpage.